Abstract

When introducing compiler optimizations, one has to make sure the meaning of the program is not changed. In particular, optimizations of recursive let-bindings should not alter the termination behaviour of an expression. During the proof thereof, unification problems between multisets of variable-to-variable bindings, i.e. equations of the form $[a_1 = b_1, ..., a_n = b_n] = [x_1 = y_1, ..., x_m = y_m]$, arise, where a_i, b_i, x_i, y_i can be either fixed or be instantiable by such fixed variable names. The problem is shown to be NP-complete, and an algorithm to find the set of all solutions is proven correct. As an extension to the problem, set variables are introduced, standing for arbitrary multisets of bindings, which also occur in optimization rules. As the decidability of the extension was previously unknown, algorithms were successively developed, starting with restricted cases such as allowing only one set variable per equation. Eventually, a concrete solution to the unrestricted multiset extension is provided, showing the problem to be decidable. Each of the extensions are proven correct, and the final, unrestricted problem is implemented in Haskell and tested extensively. Another extension is that of chain variables, representing chains of bindings of the form $[x_1 = x_2, x_2 = x_3, ..., x_{n-1} = x_n]$. A restricted case of this is being inspected.

Contents

1 Motivation

There are several ways to define the semantics of a programming language. A common way is *operational semantics*, where concrete rewriting rules specify the meaning of constructs in the language. For example, the evaluation of an if-then-else clause might be specified in one of the following ways (examples drawn from [7]):

$$\texttt{if True then } t_2 \texttt{ else } t_3 \;\rightarrow\; t_2 \qquad\qquad \texttt{if False then } t_2 \texttt{ else } t_3 \;\rightarrow\; t_3$$

$$\frac{t_1 \rightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \;\rightarrow\; \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

Figure 1.1: if-then-else defined in small-step operational semantics

$$\frac{t_1 \Downarrow \texttt{True} \qquad t_2 \Downarrow v_2}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \;\Downarrow\; v_2} \qquad\qquad \frac{t_1 \Downarrow \texttt{False} \qquad t_3 \Downarrow v_3}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \;\Downarrow\; v_3}$$

Figure 1.2: if-then-else defined in big-step operational semantics

The compiler's main task, then, (among other things like memory management and synchronization) is to implement these rules (reductions), which, however, do not have to be obeyed word-for-word. It is perfectly valid — and common — for a compiler to spot inefficiencies in advance and to remove them.

In lazily evaluated functional languages such as Haskell, where order of evaluation is irrelevant, there is plenty of room for such compiler optimizations. These optimizations are defined in terms of *optimization rules*, similar to these in operational semantics. Figure 1.3 shows an example of such a rule which GHC, Haskell's standard compiler, performs (introduced in [5]).

$$\texttt{let } x = \;\texttt{let } v{=}e \texttt{ in } b \;\texttt{ in } c \;\longrightarrow\; \texttt{let } v{=}e \texttt{ in let } x{=}b \texttt{ in } c$$

Figure 1.3: An example of an optimization rule: Let-floating

When introducing optimizations, however, one has to make sure the meaning of the program is not changed. One of the properties to be checked is *convergence equivalence*, i.e. a program's termination behaviour should not change after an optimization has taken place. In particular, assuming that a program P_0 terminates, its optimized version Q_0 should also terminate.

In a lazily evaluated language, the assumption is equivalent to P_0 evaluating to weak-head normal form (WHNF). In other words, there exists a finite sequence of reductions $P_i \to P_{i+1}$ starting at P_0 and ending in P_n, where P_n is in WHNF.

The termination of Q_0 is then shown by induction over the reduction of P_0: First, prove that for any i, if P_i is in WHNF, Q_i is also in WHNF. Then, it suffices to show that if P_i optimizes to Q_i and P_i reduces to P_{i+1}, there exists a reduction $Q_i \to Q_{i+1}$ such that P_{i+1} optimizes to Q_{i+1}. This procedure is illustrated in Figure 1.4.

$$\begin{tikzcd}
P_i \arrow[r, "\mathrm{opt}"] \arrow[d, "\mathrm{red}"'] & Q_i \arrow[d, dotted, "\mathrm{red}'"] \\
P_{i+1} \arrow[r, dotted, "\mathrm{opt}'"] \arrow[d, "*"'] & Q_{i+1} \arrow[d, dotted, "*"'] \\
P_n \arrow[r, dotted] & Q_n
\end{tikzcd}$$

Figure 1.4: The diagram method [9] for the proof of convergence equivalence

The need for unification arises at the top left corner of the diagram. Given a reduction rule $\mathrm{red} : R \to R'$ and an optimization rule $\mathrm{opt} : Q \to Q'$, if R and Q are not unifiable, the correctness of opt is irrelevant for red. Contrarily, if R and Q are unifiable by a substitution σ, we want to express $P_i := \sigma(R) = \sigma(Q)$, apply the reduction and optimization on it to obtain $P_{i+1} = \sigma(R')$ and $Q_i = \sigma(Q')$, and proceed to find red$'$ and opt$'$ to provide us with the same $Q_{i+1} = \mathrm{red}'(Q_i) = \mathrm{opt}'(P_{i+1})$.

As we do not want to be overwhelmed by the abundance of constructs to be considered in a full-blown language, we concentrate on recursive let-bindings (`letrec`). These are of the form `let` E `in` e where E consists of variable bindings of the form $x = y$, stating that x is *bound* to y. Then, x can be used in the expression e as a substitute for y. For example, `let x = 5; y = x in  x * 2 + y` is the same as `5 * 2 + 5`. The bindings can be recursive, i.e. self-containing like `f = (\x -> if x <= 1 then 1 else f (x-1) * x)`, in which case the binding does not act as a description of a mere substitute, but as a representation of the fixed point of `f`.

To study these expressions, [10] extends the call-by-need lambda calculus by `letrecs` to the calculus L_{need}. Together come reduction and transformation rules using set variables, standing for arbitrary multisets of bindings, as well as chain variables, standing for chains of bindings like `a=b,b=c,c=d`.

In the following chapters, the focus of our attention will lie on the unification of these sets or multisets of bindings that occur in the rules for the letrec expressions.

Related Work

Motivated by automatic theorem proving, general unification algorithms have been studied since [8], allowing us to unify first-order logic terms, which were later improved to yield results within linear time [6, 3]. Concretely, given a set of pairs of terms, the algorithm finds the set of substitutions that, for all pairs in the input set, makes both sides of the pair equal. For example (from [6]), a pair might look like $\langle F(x_1, x_2), F(G(x_2), G(x_3))\rangle$, which is unified by the substitution $\{x_1 \mapsto G(G(x_3)), x_2 \mapsto G(x_3)\}$, giving us the pair $\langle F(G(G(x_3)), G(x_3)), F(G(G(x_3)), G(x_3))\rangle$ when applied. Our set-up $[a_1 = b_1, ..., a_n = b_n]$ of letrec bindings could also be described as the first-order term $E_n(B(a_1, b_1), ..., B(a_n, b_n))$. However, this representation would not account for the commutativity of E, or in other words, the expression would be seen as a list of bindings instead of a multiset. [2] provides ways to deal with exactly this (among with a few other data structures); furthermore, it incorporates second-order "tails", or single set variables, in our nomenclature. [4] extends this to allow multiple set variables. Parts of this thesis are special cases of these results (cf. next subsection).

Overview

In Chapter 2, basic concepts like expressions and substitutions will be introduced, followed by a definition and solution of the simple unification problem, i.e. sets of equations of the form $[a_1 = b_1, ..., a_n = b_n] = [x_1 = y_1, ..., x_m = y_m]$. In Section 2.3, the problem is shown to be NP-complete.

In Chapter 3, we will study extensions to the simple problem, where Sections 3.1, 3.3 and 3.4 will cover multiset extensions. In Section 3.1, a single multiset variable will be allowed on only one side of each equation. In Section 3.3, both sides of the equation will be allowed to have at most one multiset variable (this is a special case of "bags with tails" discussed in [2]). Section 3.2 is a variant of this, where the expressions as well as their variables are considered true sets instead of multisets. In Section 3.4, all restrictions will be lifted, such that an arbitrary number of multisets is allowed to appear anywhere in the problem. A special case of this is mentioned in Subsection 3.4.4 in which the variables linear, i.e. each variable can appear only once in the whole problem. This is a special case of [4]. The extension in Section 3.5 allows a single chain variable — a new type of multiset variable — on one side of each equation, which can only be instantiated by an expression which is of the form $[x_0 = x_1, x_1 = x_2, ..., x_{n-1} = x_n]$.

In Chapter 4, the functionalities (Section 4.1) of the implementation of the algorithm from Section 3.4 (which includes both of the problems from Sections 3.1 and 3.3) will be presented. Section 4.2 mentions acceleration rules, an implementation detail. Finally, in Section 4.3, tests conducted on the algorithm will be outlined.

2 The Simple Unification Problem for Variable Bindings

2.1 The Problem

2.1.1 Expressions

General expressions

Variable bindings are of the form $x = y$, where x and y are (not necessarily distinct) variables from some variable space V (any countable set) with $x, y \in V$. Expressions $Expr_V$ over V are finite multisets of such variable bindings.

Definition 2.1.1 (expressions)**.** *An expression of variable bindings over the variable space V is defined by the following BNF grammar:*

$$\begin{aligned} Expr_V &::= \emptyset \mid Bind_V, Expr_V && \textit{(expressions)} \\ Bind_V &::= Var_V = Var_V && \textit{(bindings)} \\ Var_V &::= V && \textit{(variables)} \end{aligned}$$

Notation. *Instead of closing every non-empty expression with a "$, \emptyset$", the whole expression may be written in list-like brackets ([]).*

Example. *Assuming* $\mathsf{x}, \mathsf{b}, \mathsf{d}, A, X, Y \in V$*, the following words are all expressions: "$\emptyset$", "$\mathsf{x} = A, \emptyset$", "$X = Y, \mathsf{b} = \mathsf{b}, \mathsf{b} = \mathsf{d}, \emptyset$"; alternatively written as "$[]$", "$[\mathsf{x} = A]$" and "$[X = Y, \mathsf{b} = \mathsf{b}, \mathsf{b} = \mathsf{d}]$", respectively.*

As forementioned, we see expressions as a multiset, and therefore ignore the order of its elements. Formally, we define:

Definition 2.1.2 (multiset-equality of expressions)**.** *Let $e_1 := [b_1, ..., b_n]$ and $e_2 := [b'_1, ..., b'_m]$ be expressions. Then, e_1 and e_2 are equal as a multiset, denoted $e_1 \sim e_2$, if and only if $n = m$ and $\exists \pi \in \mathfrak{S}_n \; \forall i \in \{1, ..., n\} : \; b_i = b'_{\pi(i)}$, where $\mathfrak{S}_n$ is the set of bijections on $\{1, ..., n\}$.*

Notation. *Henceforth, $e_1 = e_2$ stands for $e_1 \sim e_2$ unless otherwise noted.*

Definition 2.1.3. *Let $e_1 = [b_1, ..., b_n]$ and $e_2 = [b'_1, ..., b'_m]$ be expressions. Concatenation $e_1 \cup e_2$, difference $e_1 \setminus e_2$ and the subset relation $e_1 \subseteq e_2$ are defined in their usual ways.*

Lemma 2.1.1. *If* $A = [a_1, ..., a_n]$*,* $B = [b_1, ..., b_m]$ *and* $X = [x_1, ..., x_k]$ *are expressions,* $X \cup A = X \cup B$ *implies* $A = B$*.*

Proof. Assume $[x_1, ..., x_k, a_1, ..., a_n] = [x_1, ..., x_k, b_1, ..., b_m]$, i.e. $k + n = k + m$ and there exists a bijection $\pi : \{1, ..., k + n\} \to \{1, ..., k + n\}$ such that $x_i = x_{\pi(i)}$ (if $\pi(i) \leq k$) or $x_i = b_{\pi(i)-k}$ (if $\pi(i) > k$) for all $i \in \{1, ..., k\}$ and $a_{i-k} = x_{\pi(i)}$ or $a_{i-k} = b_{\pi(i)-k}$ for all $i \in \{k + 1, ..., k + n\}$.
We show $A = B$ by constructing a bijection $\rho : \{1, ..., n\} \to \{1, ..., n\}$ such that $a_i = b_{\rho(i)}$ for all $i \in \{1, ..., n\}$: For each $i \in \{1, ..., n\}$, there exists a smallest $c_i \in \mathbb{N}_+$ such that $j := \pi^{c_i}(k + i) > k$, as permutations are known to have finite characteristics. Choosing $\rho(i) := j - k$, one verifies that ρ is injective, which suffices for bijectivity: Assuming $\rho(i_1) = \rho(i_2)$, i.e. $\pi^{c_{i_1}}(i_1 + k) - k = \pi^{c_{i_2}}(i_2 + k) - k$ or $\pi^{c_{i_1}}(i_1 + k) = \pi^{c_{i_2}}(i_2 + k)$ after adding k to both sides, $c_{i_1} \neq c_{i_2}$ (without loss of generality, $c_{i_1} < c_{i_2}$) would result in a contradiction $i_1 + k = \pi^{c_{i_2} - c_{i_1}}(c_{i_2})$ to the minimality of c_{i_2}. Hence, $i_1 = i_2$ from the bijectivity of $\pi^{c_{i_1}} = \pi^{c_{i_2}}$. □

Furthermore, we note at this point how we can extend a map between variable spaces to operate on expressions:

Definition 2.1.4 (canonical extension)**.** *Given two variable spaces* V *and* W*, the extension operators* ω_{Bind} *and* ω_{Expr} *are defined as follows: For all* $f : V \to W$ *and* $v, v' \in V$*,*

$$\begin{aligned} \omega_{Bind} &: (V \to W) \to (Bind_V \to Bind_W) \\ &\omega_{Bind}(f)(v = v') := f(v) = f(v') \end{aligned}$$

$$\begin{aligned} \omega_{Expr} &: (V \to W) \to (Expr_V \to Expr_W) \\ &\omega_{Expr}(f)([b_1, ..., b_n]) := [\omega_{Bind}(f)(b_1), ..., \omega_{Bind}(f)(b_n)] \end{aligned}$$

Examples. *Remark 2.1, Def. 2.1.7.*

Lemma 2.1.2. *Canonical extension preserves injectivity.*

Proof. Let $f : V \to W$ be injective. Then, $\omega_{Bind}(f)$ is injective, since for any bindings $(v = v')$ and $(w = w')$,

$$\omega_{Bind}(f)(v = v') = (f(v) = f(v')) = (f(w) = f(w')) = \omega_{Bind}(f)(w = w')$$

implies $f(v) = f(w)$ and $f(v') = f(w')$ and thus, due to injectivity of f, $v = w$ and $v' = w'$ and hence $(v = v') = (w = w')$.
$\omega_{Expr}(f)$ is also injective, since for any expressions $[b_1, ..., b_n]$ and $[b'_1, ..., b'_m]$,

$$\begin{aligned} \omega_{Expr}(f)([b_1, ..., b_n]) &= [\omega_{Bind}(f)(b_1), ..., \omega_{Bind}(f)(b_n)] \\ &= [\omega_{Bind}(f)(b'_1), ..., \omega_{Bind}(f)(b'_m)] = \omega_{Expr}(f)([b'_1, ..., b'_m]) \end{aligned}$$

implies $n = m$ and the existence of a $\pi \in \mathfrak{S}_n$ such that $\forall i \in \{1, ..., n\} : \omega_{Bind}(f)(b_i) = \omega_{Bind}(f)(b'_{\pi(i)})$ and thus, due to injectivity of $\omega_{Bind}(f)$, $b_i = b'_{\pi(i)}$ for each $i \in \{1, ..., n\}$ and hence $[b_1, ..., b_n] = [b'_1, ..., b'_m]$. □

Ground expressions and expressions with metavariables

From now on we consider expressions over concrete variable spaces: ground variables, which stand for actual program variables, and metavariables, which we will use as placeholders for unknown program variables.

Definition 2.1.5 (ground variables and expressions). *We denote $G := \{\mathsf{x}_1, \mathsf{x}_2, ...\}$, or, more commonly, Var_G, the set of ground variables. Elements of Expr_G are called ground expressions.*

Definition 2.1.6 (expressions with metavariables). *Let $H := \{X_1, X_2, ...\}$ be the set of metavariables. We denote $M := \mathit{Var}_G \cup H$, or, more commonly, Var_M, the set of variables including metavariables. Elements of Expr_M are called expressions with metavariables.*

Notation. *The index for variables might be omitted, and other letters might be used. Also, we often misuse the notation to use ground and meta variables as variables standing for variables, i.e. X_1 might be a variable standing for X_2, although from the definitions above, $X_1 \neq X_2$ should always hold. Equally, x might stand for e.g. y. (Presumably, introducing a separate notation for meta-ground-variables and meta-meta-variables would be more confusing.)*

Examples. *X_{1987}, X_0, X, A, B, C_{20}, D_3 are all metavariables. x_{1987}, x_0, x, a, b, c_{20}, d_3 are all ground variables.*

Remark 2.1. The inclusion $G \subseteq_\iota M$ extends to $\mathit{Expr}_G \subseteq_{\omega_{\mathit{Expr}}(\iota)} \mathit{Expr}_M$ canonically (Lemma 2.1.2).

2.1.2 Substitutions

We now make the notion of "metavariables as placeholders for ground variables" more concise by introducing substitutions: functions that replace finitely many metavariables by other variables. A special case of these are ground substitutions that exclude metavariables from its image.

Definition 2.1.7 (substitutions). *A substitution is a function $\sigma : \mathit{Var}_M \to \mathit{Var}_M$, as well as its canonical extensions $\omega_{\mathit{Bind}}(\sigma)$ and $\omega_{\mathit{Expr}}(\sigma)$, such that the restriction $\sigma|_{\mathit{Var}_G}$ to the subset of program variables is the identity function $\mathrm{id}|_{\mathit{Var}_G}$ and there exists a finite set F such that $\sigma|_{\mathit{Var}_M \setminus F}$ is also $\mathrm{id}|_{\mathit{Var}_M \setminus F}$. The set of all ground substitutions is denoted Subst_M.*

Definition 2.1.8 (ground substitutions). *A ground substitution is a substitution σ such that $\mathrm{im}(\sigma) \subseteq \mathit{Var}_G$. The set of all ground substitutions is denoted Subst_G.*

Notation. *Where it is clear from the context, ω is omitted and one simply writes $\sigma(v = v')$ or $\sigma([b_1, ..., b_n])$.*

Notation. *As substitutions leave all but finitely many variables the same, one can write any substitution σ as $\{X_{\rho(1)} \mapsto \sigma(X_{\rho(1)}), ..., X_{\rho(n)} \mapsto \sigma(X_{\rho(n)})\}$ for some $n \in \mathbb{N}$, where $\rho : \{1, ..., n\} \to \mathbb{N}$ is injective and $\sigma(X_i) = X_i$ for all $i \in \mathbb{N} \setminus \mathrm{im}(\rho)$.*

Examples. *$\sigma := \{X \mapsto \mathsf{y}, A \mapsto B\}$ is a substitution, with e.g.*

$$\omega_{Expr}(\sigma)([X = \mathsf{a}]) = [\omega_{Bind}(\sigma)(X = \mathsf{a})] = [\sigma(X) = \sigma(\mathsf{a})] = [\mathsf{y} = \mathrm{id}(\mathsf{a})] = [\mathsf{y} = \mathsf{a}],$$

or $\sigma([X = \mathsf{a}]) = [\mathsf{y} = \mathsf{a}]$, for short. $\{X \mapsto \mathsf{a}, X \mapsto B\}$ is not a substitution, since it is ill-defined (ρ is not injective). $X_i \mapsto X_{i+1}$ is also not a substitution, since it changes infinitely many variables.

Composition and Generality

Notation (composition of substitutions)**.** *The composition, i.e. successive application, of multiple substitutions $\sigma_1, ..., \sigma_n$ is written with $\circ$ and from right to left: $\sigma_n \circ ... \circ \sigma_1$. We might omit $\circ$ where appropriate.*

Remark 2.2. Let $a = \{A_1 \mapsto a_1, ..., A_n \mapsto a_n\}$ and $b = \{B_1 \mapsto b_1, ..., B_m \mapsto b_m\}$, where there might be i and j such that $A_i = B_j$. Let $\{B'_1, ..., B'_s\} = \{B_1, ..., B_m\} \setminus \{A_1, ..., A_n\}$ and $b'_i := b(B'_i)$ for each $i \in \{1, ..., s\}$. Then, $b \circ a = \{A_1 \mapsto b(a_1), ..., A_n \mapsto b(a_n), B'_1 \mapsto b'_1, ..., B'_s \mapsto b'_s\}$.

Proof. One verifies that $b \circ a$ as described above matches successive application in all of the possible cases for X: If $X = A_i$ for some i, then, $b(a(X)) = b(a(A_i)) = b(a_i)$, justifying the $(A_i \mapsto a_i)$-components. If $X \notin \{A_1, ..., A_n\}$ but $X = B_i$ for some i, by definition of $\{B'_1, ..., B'_s\}$ there must exist some j such that $X = B'_j$, hence $b(a(X)) = b(a(B'_j)) = b(B'_j) = b'_j$, as needed for $B'_i \mapsto b'_i$. If $X \notin \{A_1, ..., A_n\}$ and $X \notin \{B_1, ..., B_s\}$, then, $b(a(X)) = X$, which we can ignore according to our notational conventions. □

Definition 2.1.9 (generality and equivalence of substitutions)**.** *A substitution σ is more general than a substitution τ, denoted $\sigma \leq \tau$, when there exists a substitution λ such that $\lambda\sigma = \tau$. Two substitutions σ and τ are called equivalent if and only if $\sigma \leq \tau$ and $\tau \leq \sigma$.*

Definition 2.1.10 (restriction of the domain)**.** *One might restrict notions regarding substitutions to a subset W of the actual domain, denoted with a subscript $[W]$. For example, $\sigma =_{[W]} \tau$ if $\sigma(w) = \tau(w)$ for any $w \in W$, or $\sigma \leq_{[W]} \tau$, when there exists a substitution λ such that $\lambda\sigma =_{[W]} \tau$.*

2.1.3 Unification problems

Definition 2.1.11. *An element of a unification problem is defined by*

$$\mathit{ProbEl} ::= \mathit{Expr}_M \doteq \mathit{Expr}_M.$$

A unification problem UnifProb is a finite set of ProbEls.

Definition 2.1.12 (unifier)**.** *A unifier to a unification problem* Γ *is a substitution* Sol *such that* $\forall e_1 \doteq e_2 \in \Gamma : Sol(e_1) = Sol(e_2)$.

Definition 2.1.13 (solution)**.** *A solution to a unification problem* Γ *is a unifier* Sol *such that* Sol *is a ground substitution.*

2.2 Solution

2.2.1 Data structure

To solve the unification problem, i.e. to find some or all of the substitutions that are solutions to the problem, we use the data structure *Solver*, a finite set of *SolverEl*s, defined by:

$$\mathit{SolverEl} ::= \mathit{Expr}_M \stackrel{?}{=} \mathit{Expr}_M \mid \mathit{Bind}_M \stackrel{?}{=} \mathit{Bind}_M \mid \mathit{Var}_M \stackrel{?}{=} \mathit{Var}_M$$

Before beginning the algorithm, the problem is translated into the solver data structure by $\Upsilon : \mathit{UnifProb} \to \mathit{Solver}$, mapping $e_1 \doteq e_2 \mapsto e_1 \stackrel{?}{=} e_2$ onto the set. The notion of unifiers is analogously translated into the solver data structure.

2.2.2 Algorithm

A subset of all of the solutions to a simple unification problem $\Delta \in \mathit{UnifProb}$ is obtained by the algorithm shown in Figure 2.1. It is initialized with $(\mathrm{id}, \Upsilon(\Delta))$ and, for each state (σ, Γ) the algorithm reaches, if P holds and a rule

$$\frac{(\sigma, \Gamma)}{(\sigma_1, \Gamma_1) \mid ... \mid (\sigma_n, \Gamma_n)} P$$

exists, the algorithm transitions into some or all of the states (σ_1, Γ_1), ..., (σ_n, Γ_n) "non-deterministically" (e.g. a depth-first search, in practice), depending on whether we want only some or all of the solutions to the problem. Each branch of the algorithm terminates with either $(Sol, \emptyset)$ or *Fail*, where, in the former case, Sol is a unifier to Δ, and in the latter, it is indicated that no solutions for the correspondent branch were found.

We view this unifier as an encoding for the set of all solutions τ such that $Sol \leq \tau$, since representing all of them explicitly would require an infinite data structure.

Notation. *When we write* $\Delta \cup \Gamma$ *in a rule, we actually mean* $\Delta \dot{\cup} \Gamma$*, i.e.* $\Delta \cap \Gamma = \emptyset$*. Also,* x *(note the difference to* x *and* X*) stands for either a program variable or a metavariable.*

Notation. $\Gamma[a/b]$ *stands for* Γ *with all occurrences of* b *substituted by* a*.*

V-Tautology
$$\frac{(Sol, \{x \stackrel{?}{=} x\} \cup \Gamma)}{(Sol, \Gamma)}$$

V-Clash
$$\frac{(Sol, \{\mathsf{x} \stackrel{?}{=} \mathsf{y}\} \cup \Gamma)}{\mathit{Fail}}\ \mathsf{x} \neq \mathsf{y}$$

V-Application
$$\frac{(Sol, \{X \stackrel{?}{=} x\} \cup \Gamma)}{(\{X \mapsto x\} \circ Sol, \Gamma[x/X])}$$

V-Orientation
$$\frac{(Sol, \{\mathsf{x} \stackrel{?}{=} X\} \cup \Gamma)}{(Sol, \{X \stackrel{?}{=} \mathsf{x}\} \cup \Gamma)}$$

B-Decomposition
$$\frac{(Sol, \{x = y \stackrel{?}{=} x' = y'\} \cup \Gamma)}{(Sol, \{x \stackrel{?}{=} x', y \stackrel{?}{=} y'\} \cup \Gamma)}$$

E-Tautology
$$\frac{(Sol, \{\emptyset \stackrel{?}{=} \emptyset\} \cup \Gamma)}{(Sol, \Gamma)}$$

E-ClashR
$$\frac{(Sol, \{[b_1, ..., b_k] \stackrel{?}{=} \emptyset\} \cup \Gamma)}{\mathit{Fail}}\ k > 0$$

E-ClashL
$$\frac{(Sol, \{\emptyset \stackrel{?}{=} [b_1, ..., b_k]\} \cup \Gamma)}{\mathit{Fail}}\ k > 0$$

E-Distribution
$$\frac{(Sol, \{[b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma)}{|_{i=1}^{m}\ (Sol, \{b_1 \stackrel{?}{=} b'_i, [b_2, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma)}\ k > 0,\ m > 0$$

Figure 2.1: Algorithm for the simple unification problem

2.2.3 Correctness

Termination

We show that for any correct input, the output of the algorithm is defined. In particular, the algorithm should neither "get stuck" such that there exists no rule that is applicable, nor be able to apply rules infinitely many times.

Proposition 2.2.1. *For any SolverEl, there exists an applicable rule.*

Proof. For equations $Var_M \stackrel{?}{=} Var_M$, if the variable on the left-hand side is (syntactically) equal to the variable on the right-hand side, V-Tautology is applied. The remaining cases are covered in the following table:

	ground (R)	meta (R)
ground (L)	V-Clash	V-Orientation
meta (L)	V-Application	

Any element of the form $Bind_M \stackrel{?}{=} Bind_M$ is treated by B-Decomposition. For $Expr_M \stackrel{?}{=} Expr_M$, the following table covers every possible case:

	empty (R)	non-empty (R)
empty (L)	E-Tautology	E-ClashL
non-empty (L)	E-ClashR	E-Distribution

□

Proposition 2.2.2. *Applying the rules, Γ will eventually become empty within finitely many steps.*

Proof. A clash-rule (V-Clash, R-ClashL, E-ClashR) terminates the current branch of the algorithm immediately. For the other rules, let μ_T be the number of Ts in a *Solver* data structure, where $T \in \{ExprE,\ BindE,\ VarE,\ RMVar,\ BExpr\}$ and *ExprE* denotes *Expr*-equations, *BindE* denotes *Bind*-equations, *VarE* denotes *Var*-equations, *RMVar* denotes metavariables on the right hand side of an equation, *BExpr* the bindings inside of expressions. Formally:

$$\begin{aligned}
\kappa_{T\in\{ExprE,\ BindE,\ VarE\}} : SolverEl &\longrightarrow \mathbb{N}_0 \\
t_1 \stackrel{?}{=} t_2 &\longmapsto \begin{cases} 1, & t_1 \stackrel{?}{=} t_2 \in T \\ 0, & \text{otherwise} \end{cases} \\
\kappa_{RMVar} : SolverEl &\longrightarrow \mathbb{N}_0 \\
v \stackrel{?}{=} X_i &\longmapsto 1 \\
t_1 \stackrel{?}{=} t_2 &\longmapsto 0,\ t_1, t_2 \notin Var \text{ or } t_2 \text{ is not meta}
\end{aligned}$$

$$\begin{aligned} \kappa_{BExpr} : \mathit{SolverEl} &\longrightarrow \mathbb{N}_0 \\ [b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m] &\longmapsto k + m \\ t_1 \stackrel{?}{=} t_2 &\longmapsto 0,\ t_1, t_2 \notin \mathit{Expr} \end{aligned}$$

and for every $T \in \{\mathit{ExprE},\ \mathit{BindE},\ \mathit{VarE},\ \mathit{RMVar},\ \mathit{BExpr}\}$:

$$\begin{aligned} \mu_T : \mathit{Solver} &\longrightarrow \mathbb{N}_0 \\ \Gamma &\longmapsto \sum_{\gamma \in \Gamma} \kappa_T(\gamma) \end{aligned}$$

Then, for each rule application, the measure

$$\begin{aligned} \mu : \mathit{Solver} &\longrightarrow \mathbb{N}_0^5 \\ \Gamma &\longmapsto (\mu_{\mathit{ExprE}}(\Gamma), \mu_{\mathit{BExpr}}(\Gamma), \mu_{\mathit{BindE}}(\Gamma), \mu_{\mathit{VarE}}(\Gamma), \mu_{\mathit{RMVar}}(\Gamma)) \end{aligned}$$

strictly decreases with respect to the lexicographic ordering on $\mathbb{N}_0^5$. Eventually, the measure must reach $(0, ..., 0)$ in which case Γ is empty.

Indeed, V-Tautology decreases μ_{VarE} by 1. If x is a metavariable, μ_{RMVar} also decreases by 1. The other measures stay the same. Similarly, μ_{ExprE} decreases by 1 on application of E-Tautology whereas the other measures do not change. V-Application decreases μ_{VarE} by at least 1 as the term $\{X \stackrel{?}{=} x\}$ is discarded. μ_{VarE} can, along with μ_{BExpr}, μ_{ExprE}, μ_{BindE} and μ_{RMVar}, decrease by more than one, if the rewriting of X to x causes previously distinct elements of Γ to become equal. Additionally, if x was a ground variable, μ_{RMVar} can decrease by up to the number of times X appeared on the right-hand side. The application of V-Orientation decreases μ_{RMVar} by 1, and, if $X \stackrel{?}{=} \mathsf{x}$ was already contained in Γ, μ_{VarE} also decreases by 1. The other measures retain their values. B-Decomposition increases μ_{VarE} and μ_{RMVar} by at most 2, but decreases μ_{BindE} by 1. For each branch, E-Distribution decreases μ_{BExpr} at least by 2, compensating the increase of at most 1 in μ_{BindE}.

Table 2.1 provides an overview for the changes in the measures, where "dec" stands for decrease and "inc" for increase, "?" indicating that the change is possible but not necessary.

Rule	ExprE	BExpr	BindE	VarE	RMVar
V-Taut	-	-	-	**dec**	dec?
V-App	dec?	dec?	dec?	**dec**	dec?
V-Ori	-	-	-	dec?	**dec**
B-Dec	-	-	**dec**	inc?	inc?
E-Taut	**dec**	-	-	-	-
E-Dist	-	**dec**	inc?	-	-

Table 2.1: Measure changes during the simple problem

□

Soundness

We have to show that all results the algorithm yields is a valid unifier to the unification problem. To this end, it suffices to prove the following invariant:

Theorem 2.2.1. *For each interim result* (σ, Γ) *and for each substitution* τ *unifying* Γ, $\tau\sigma$ *is a unifier to* Γ_{ini}*, the initial problem.*

Then, when the algorithm terminates with $(\sigma, \emptyset)$, since id unifies $\emptyset$, $\mathrm{id} \circ \sigma = \sigma$ is a unifier to the unification problem.

Proof. We verify that the invariant holds for the initialization $(\mathrm{id}, \Gamma_{ini})$: if τ is a unifier for Γ_{ini}, then, $\tau \circ \mathrm{id}$ also unifies Γ_{ini}.
For the clash-rules (V-Clash, R-ClashL, E-ClashR), there is nothing to show, since they do not yield any result. For each other rule $\dfrac{(\sigma, \Gamma)}{(\sigma', \Gamma')} P$, we assume that P holds and, for all τ unifying Γ, $\tau\sigma$ is a unifier, and show that for all τ' unifying Γ', $\tau'\sigma'$ is also a unifier.
V-Tautology: Let τ' be a unifier for Γ. Then, since $\{x \stackrel{?}{=} x\}$ is already unified, $\tau := \tau'$ is also a unifier for $\{x \stackrel{?}{=} x\} \cup \Gamma$. Hence, $\tau \circ \mathit{Sol} = \tau' \circ \mathit{Sol}$ is a unifier.
E-Tautology: Analogous to V-Tautology.
V-Application: Let τ' be a unifier for $\Gamma[x/X]$. Then $\tau := \tau' \circ \{X \mapsto x\}$ unifies $\{X \stackrel{?}{=} x\} \cup \Gamma$, since $\tau(\{X \stackrel{?}{=} x\} \cup \Gamma) = \tau'(\{x \stackrel{?}{=} x\} \cup \Gamma[x/X])$ and $\{x \stackrel{?}{=} x\}$ is already unified. Hence, $\tau \circ \mathit{Sol} = \tau' \circ \{X \mapsto x\} \circ \mathit{Sol}$ is a unifier.
V-Orientation: Let τ' be a unifier for $\{X \stackrel{?}{=} \mathsf{x}\} \cup \Gamma$. Then, $\tau'(X)$ must be x, which makes $\tau := \tau'$ also a unifier of $\{\mathsf{x} \stackrel{?}{=} X\} \cup \Gamma$. Hence, $\tau \circ \mathit{Sol} = \tau' \circ \mathit{Sol}$ is a unifier.
B-Decomposition: Let τ' be a unifier for $\{x \stackrel{?}{=} x', y \stackrel{?}{=} y'\} \cup \Gamma$. Then, $\tau'(x) = \tau'(x')$ and $\tau'(y) = \tau'(y')$ must hold, which makes $\tau := \tau'$ also a unifier of $\{x = y \stackrel{?}{=} x' = y'\} \cup \Gamma$, since $\tau'(x = y) = (\tau'(x) = \tau'(y)) = (\tau'(x') = \tau'(y')) = \tau'(x' = y')$. Hence, $\tau \circ \mathit{Sol} = \tau' \circ \mathit{Sol}$ is a unifier.
E-Distribution: Let $k > 0$, $m > 0$ and τ' be a unifier for $\{b_1 \stackrel{?}{=} b'_i, [b_2, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$ for some $i \in \{1, ..., k\}$, i.e. $\tau'(b_1) = \tau'(b'_i)$ and $\tau'([b_2, ..., b_k]) = \tau'([b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m])$. Then, there exists a permutation $\pi \in \mathfrak{S}_k$ such that $\tau'(b_j) = \tau'(b_{\pi(j)})$ (with $\pi(1) = i$). This makes $\tau := \tau'$ also a unifier of $\{[b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma$. Hence, $\tau \circ \mathit{Sol} = \tau' \circ \mathit{Sol}$ is a unifier. □

Completeness

Lemma 2.2.1. *For each interim result* (σ, Γ)*, if there exists an equation in* Γ *that contains an* $X \in \mathit{Var}_M$*, then,* $\sigma(X) = X$.

Proof. By induction on the structure of the derivation of the interim result. Clearly, the lemma holds for the initialization $(\mathrm{id}, \Gamma_{ini})$. One sees that there is no rule that adds metavariables to Γ. Regarding σ, V-Application is the only rule that changes it.

Assuming *Sol* does not change metavariables occurring in $\{X \stackrel{?}{=} x\} \cup \Gamma$, $\{X \mapsto x\} \circ Sol$ also does not change metavariables in $\Gamma[x/X]$, since the metavariable X was substituted away. □

We have to make sure that there is no solution to the unification problem that is not found by the algorithm. To this end, it suffices to prove the following invariant:

Theorem 2.2.2. *For each interim result* (σ, Γ) *and for each solution* η *of the initial problem* Γ_{ini}*, there exists a solution* τ *to* Γ *such that* $\eta = \tau\sigma$*.*

Then, there must exist some conclusion $(\sigma, \emptyset)$ and a substitution τ such that $\eta = \tau\sigma$, specifically, $\eta \geq \sigma$, showing that every solution η is taken into account by a representative lower bound.

Proof. The invariant holds for the initialization $(\mathrm{id}, \Gamma_{ini})$: if η is a solution to Γ_{ini}, then there exists a solution $\tau := \eta$ to Γ_{ini} such that $\eta = \eta \circ \mathrm{id}$.
For each rule $\dfrac{(\sigma, \Gamma)}{|_{i=1}^{n}\ (\sigma_i', \Gamma_i')} P$, assuming that P holds and there exists a substitution τ such that $\eta := \tau\sigma$ is a solution for Γ, we show that there exists an $i \in \{1, ..., n\}$ and a substitution τ' such that $\eta = \tau'\sigma_i'$ and η is a solution for Γ_i'.
For the clash rules (V-Clash, E-ClashR, E-ClashL), observe that the premises ($\{\mathsf{x} \stackrel{?}{=} \mathsf{y}\}$ with $\mathsf{x} \neq \mathsf{y}$ and $\{[b_1, ..., b_k] \stackrel{?}{=} \emptyset\}$, $\{\emptyset \stackrel{?}{=} [b_1, ..., b_k]\}$ with $k > 0$) can never be unified, making the statement hold trivially.
V-Application: Assume $\tau \circ Sol$ is a solution for $\{X \stackrel{?}{=} x\} \cup \Gamma$, in particular, $\tau(X) = \tau \circ Sol(X) = \tau \circ Sol(x) = \tau(x)$ (*Sol* can be removed due to Lemma 2.2.1). Then, $\tau \circ \{X \mapsto x\} \circ Sol = \eta = \tau \circ Sol$ as needed, since on both side of the equation, X, as the only variable the change has an effect, is sent to $\tau(x)$. It remains to show that η also solves $\Gamma[x/X]$, which holds since the substitution solved Γ already and $[x/X]$ makes no difference as η, also a solution to $\{X \stackrel{?}{=} x\}$, did not distingish between x and X anyway.
V-Tautology: Assume η solves $\{x \stackrel{?}{=} x\} \cup \Gamma$. Then, it also must have solved the subset Γ.
E-Tautology: Analogous to V-Tautology.
V-Orientation: Assume η solves $\{\mathsf{x} \stackrel{?}{=} X\} \cup \Gamma$, i.e. it solves Γ and $\eta(X) = \mathsf{x}$. Then it solves $\{X \stackrel{?}{=} \mathsf{x}\}$ and Γ, thus η is a solution to $\{X \stackrel{?}{=} \mathsf{x}\} \cup \Gamma$.
V-Decomposition: Assume η solves $\{x = y \stackrel{?}{=} x' = y'\} \cup \Gamma$, i.e. it solves Γ and $\eta(x = y) = (\eta(x) = \eta(y)) = (\eta(x') = \eta(y')) = \eta(x' = y')$. Then, $\eta(x) = \eta(x')$ and $\eta(y) = \eta(y')$, making η also a solution to $\{x \stackrel{?}{=} x', y \stackrel{?}{=} y'\} \cup \Gamma$.
E-Distribution: Assume $k > 0$ and $m > 0$ and that η solves $\{[b_1, ..., b_k] \stackrel{?}{=} [b_1', ..., b_m']\} \cup \Gamma$. Specifically, $\eta([b_1, ..., b_k]) = [\eta(b_1), ..., \eta(b_k)] = [\eta(b_1'), ..., \eta(b_m')] = \eta([b_1', ..., b_m'])$, i.e. $k = m$ and there exists a permutation $\pi \in \mathfrak{S}_k$ such that $\eta(b_1) = \eta(b'_{\pi(1)}), ..., \eta(b_k) = \eta(b'_{\pi(k)})$. Choosing $i := \pi(1) \in \{1, ..., k\}$, η also solves $\{b_1 \stackrel{?}{=} b_i', [b_2, ..., b_k] \stackrel{?}{=} [b_1', ..., b_{i-1}', b_{i+1}', ..., b_m']\} \cup \Gamma$. □

2.3 Complexity

Theorem 2.3.1. *The simple unification problem is NP-complete.*

Proof. Given a problem P, the finite sets M and G of all meta and ground variables occurring in P can be found in polynomial time. Let M' denote a set of ground variables not occurring in G and of size $|M|$. For each metavariable, a non-deterministic machine either guesses the correct ground variable out of $M' \cup G$ (which is finite) to send the meta variable to, or fails to find such a variable to conclude the emptiness of the solution set. One verifies that this covers all the possibilities, since any solution has at most $|M|$ fresh variables in its image, where their concrete naming is irrelevant. This provides us with a solution or a failure in non-deterministic linear time, showing that the simple unification problem lies in NP.

Now, let $F := f(X_1, ..., X_n)$ be a boolean formula over the variable space $X_1, ..., X_n$ in conjunctive normal form, where each disjunction has at most 3 variables. The satisfiability of F, also known as 3-SAT, is known to be to be NP-hard, which we reduce to a unification problem P. For each variable X_i, $i \in \{1..n\}$, we identify $X_{n+i} := \neg X_i$ and encode this by adding $[X_i = X_{n+i}, X_{n+i} = X_i] \doteq [\mathsf{t} = \mathsf{f}, \mathsf{f} = \mathsf{t}]$ to P. Each disjunction is of the form $X_i \vee X_j \vee X_k$, $X_i \vee X_j$, or X_i, for which we add $[\mathsf{v} = X_i, \mathsf{v} = X_j, \mathsf{v} = X_k] \doteq [\mathsf{v} = \mathsf{t}, \mathsf{v} = A', \mathsf{v} = A'']$, $[\mathsf{v} = X_i, \mathsf{v} = X_j] \doteq [\mathsf{v} = \mathsf{t}, \mathsf{v} = A']$, or $[\mathsf{v} = X_i] \doteq [\mathsf{v} = \mathsf{t}]$, respectively, where A' and A'' occur only once in the whole problem.

Then, F is satisfiable, if and only if P has a solution: If P has a solution, applying it satisfies F, as each conjunction contains a meta variable which is sent to t. If F is satisfiable by a mapping $b : \{X_1, ..., X_n\} \to \{\mathsf{t}, \mathsf{f}\}$, this is a solution, in particular, for each conjunction in F, at least one of the variable is mapped to t. $\square$

3 Multiset Extensions

3.1 Single multiset on one side

3.1.1 Problem statement

We now extend the simple unification problem by a new type of variable: multiset variables, which stand for an unknown number of bindings.

Definition 3.1.1 (multiset variables)**.** *We denote* $MSet = \{M_0, M_1, M_2, ...\}$*, or, more commonly,* Var_{MSet}*, the set of multiset variables. Elements of* Var_{MSet} *are called multiset variables.*

To begin with, each expression is only allowed to have at most one multiset variable.

Definition 3.1.2. *(single multiset expressions) An expression of variable bindings with single multiset variables over the variable space* V *is defined by the following BNF grammar:*

$$Expr_V^{SM} ::= Expr_V \mid Var_{MSet} : Expr_V$$

In addition to Definition 2.1.4, we need a new canonical extension for variable substitutions:

Definition 3.1.3. *(canonical extensions to single multiset expressions) Given two variable spaces* V *and* W*, the extension operator* $\omega_{Expr^{SM}}$ *is defined for all* $f : V \to W$ *and* $e \in Expr$*:*

$$\omega_{Expr^{SM}} : (V \to W) \to (Expr_V^{SM} \to Expr_W^{SM})$$
$$\omega_{Expr^{SM}}(f)(e) = \begin{cases} M : \omega_{Expr}(f)(e'), & e = M : e' \\ \omega_{Expr}(f)(e), & otherwise \end{cases}$$

Furthermore, we need a new type of substitution that substitutes a multiset variable with an expression:

Definition 3.1.4 (multiset substitutions and its application)**.** *Multiset substitutions are functions* $\sigma : Var_{MSet} \to Expr_V^{SM}$ *such that there exists a finite set* F *where* $\sigma\mid_{MSet \setminus F}$ *is the constant mapping to* $\emptyset$*.*

The following defines the application operator ξ for all multiset substitutions f, which might be omitted wherever appropriate:

$$\begin{aligned}
\xi : (\mathit{Var}_{MSet} \to \mathit{Expr}^{SM}_{V}) &\to (\mathit{Expr}^{SM}_{V} \to \mathit{Expr}^{SM}_{V}) \\
\xi(f)(M : e) &= \begin{cases} e' \cup e, f(M) = e' \\ M' : (e' \cup e), f(M) = M' : e' \end{cases} \\
\xi(f)(e) &= e
\end{aligned}$$

To further simplify the problem, we restrict equations to have only one multiset variable at most.

Definition 3.1.5. *An element of a unification problem one-sidedly extended by multisets is defined by*

$$\mathit{ProbEl}^{USM} ::= \mathit{Expr}_M \doteq \mathit{Expr}^{SM}_{M} \mid \mathit{Expr}^{SM}_{M} \doteq \mathit{Expr}_M$$

A unification problem $\mathit{UnifProb}^{USM}$ is a finite set of ProbEl^{USM}s.

3.1.2 Solution

$\mathit{SolverEl}$s now use Expr^{SM}_{M}s instead of Expr_Ms:

$$\mathit{SolverEl}^{SM} ::= \mathit{Expr}^{SM}_{M} \stackrel{?}{=} \mathit{Expr}^{SM}_{M} \mid \mathit{Bind}_M \stackrel{?}{=} \mathit{Bind}_M \mid \mathit{Var}_M \stackrel{?}{=} \mathit{Var}_M$$

The solution to ProbEl^{USM} is given by the rules shown in Figure 3.1 which come as an addition to our former rules from Figure 2.1.

3.1.3 Correctness

Termination

In addition to μ from the simple unification problem, we count $\kappa_{RMSetVar}$, the number of multiset variables on the right side of the equation, as well as $\kappa_{MSetVar}$ the number of multiset variables. Formally:

$$\begin{aligned}
\kappa_{RMSetVar} : \mathit{SolverEl}^{SM} &\longrightarrow \mathbb{N}_0 \\
t_1 \stackrel{?}{=} M : e &\longmapsto 1 \\
t_1 \stackrel{?}{=} t_2 &\longmapsto 0, t_2 \in \mathit{Expr}
\end{aligned}$$

$$\begin{aligned}
\kappa_{MSetVar} : \mathit{SolverEl}^{SM} &\longrightarrow \mathbb{N}_0 \\
t_1 \stackrel{?}{=} t_2 &\longmapsto \begin{cases} 1, t_1 \text{ xor } t_2 \text{ is of the form } M : e \\ 2, \text{both are of the form } M : e \\ 0, \text{otherwise} \end{cases}
\end{aligned}$$

$$\textsc{E-Set-Application}\quad \frac{(Sol, \{M : \emptyset \stackrel{?}{=} [b_1, ..., b_n]\} \cup \Gamma)}{(\{M \mapsto [b_1, ..., b_n]\} \circ Sol, \Gamma[[b_1, ..., b_n]/M])}$$

$$\textsc{E-Set-Distribution}\quad \frac{(Sol, \{M : [b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma)}{|_{i=1}^{m}\ (Sol, \{b_1 \stackrel{?}{=} b'_i, M : [b_2, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma)}\ k > 0,\ m > 0$$

$$\textsc{E-Set-Clash}\quad \frac{(Sol, \{M : [b_1, ..., b_k] \stackrel{?}{=} \emptyset\} \cup \Gamma)}{Fail}\ k > 0 \qquad \textsc{E-Set-Orientation}\quad \frac{(Sol, \{e_1 \stackrel{?}{=} M : e_2\} \cup \Gamma)}{(Sol, \{M : e_2 \stackrel{?}{=} e_1\} \cup \Gamma)}$$

Figure 3.1: single set variable on one side of the equation

and

$$\begin{aligned} \mu_{T\in\{RMSetVar, MSetVar\}} : Solver^{SM} &\longrightarrow \mathbb{N}_0 \\ \Gamma &\longmapsto \sum_{\gamma\in\Gamma} \kappa_T(\gamma) \end{aligned}$$

Then, for each rule application, the measure

$$\begin{aligned} \mu^{SM} : Solver^{SM} &\longrightarrow \mathbb{N}_0^7 \\ \Gamma &\longmapsto (\mu_{MSetVar}(\Gamma), \mu_{RMSetVar}(\Gamma), \mu(\Gamma)) \end{aligned}$$

strictly decreases with respect to the lexicographic ordering on $\mathbb{N}_0^7$.
Indeed, the rules from the simple unification problem do not change $\mu_{RMSetVar}$ or $\mu_{MSetVar}$. E-Set-Application decreases $\mu_{MSetVar}$. E-Set-Distribustion reduces μ and leaves $\mu_{MSetVar}$ as well as $\mu_{RMSetVar}$. E-Set-Clash terminates immediately. E-Set-Orientation reduces $\mu_{RMSetVar}$ and does not change $\mu_{MSetVar}$.

Rule	MSetVar	RMSetVar	ExprE	BExpr	BindE	VarE	RMVar
App	**dec**	dec?	dec	?	-	-	-
Dist	-	-	dec?	**dec**	inc?	-	-
Orient	-	**dec**	dec?	-	-	-	-

Soundness

For E-Set-Clash there is nothing to show, since it does not yield any result.
E-Set-Application: Assume that for any τ that unifies $\{M : \emptyset \stackrel{?}{=} e\} \cup \Gamma$, $\tau \circ Sol$ unifies

Γ_{ini}. Let τ' be a unifier for $\Gamma[e/M]$. To be shown is that $\tau' \circ \{M \mapsto e\} \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau' \circ \{M \mapsto e\}$, τ unifies $\{M : \emptyset \stackrel{?}{=} e\} \cup \Gamma$, since $\{M \mapsto e\}(\{M : \emptyset \stackrel{?}{=} e\} \cup \Gamma) = \{e \stackrel{?}{=} e\} \cup \Gamma[e/M]$ and τ' unifies both $\{e \stackrel{?}{=} e\}$ (already unified) and $\Gamma[e/M]$. Hence, $\tau \circ Sol = \tau' \circ \{M \mapsto e\} \circ Sol$ is a unifier for Γ_{ini}.
E-Set-Distribution: Assume $k > 0, m > 0$ and that for any τ that unifies $\{M : [b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\{b_1 \stackrel{?}{=} b'_i, M : [b_2, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$, in particular, $\tau'(b_1) = \tau'(b'_i)$ and $\tau'(M : [b_2, ..., b_k]) = \tau'([b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m])$, i.e. $\tau'(M) = [m_1, ..., m_l]$, $l + k - 1 = m - 1$ and there exists a $\pi \in \mathfrak{S}_m$ with $\pi(i) = 1$ and $\tau'(m_i) = \tau'(b_{\pi(i)})$ as well as $\tau'(b_i) = \tau'(b_{\pi(i+l)})$ for each appropriate $i \in \mathbb{N}$. This makes $\tau := \tau'$ also a unifier of $\{M : [b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma$. Hence, $\tau \circ Sol = \tau' \circ Sol$ unifies Γ_{ini}.
E-Set-Orientation Let τ' be a unifier for $\{M_2 : \emptyset \stackrel{?}{=} M_1 : e_1\} \cup \Gamma$. Then, $\tau := \tau'$ also unifies $\{M_1 : e_1 \stackrel{?}{=} M_2 : \emptyset\} \cup \Gamma$, from which one concludes that $\tau \circ Sol = \tau' \circ Sol$ unifies Γ_{ini}, as needed.

Completeness

Lemma 3.1.1. *Lemma 2.2.1 still holds.*

Proof. There is still no rule that adds metavariables to Γ. There is no additional rule changing the behaviour of σ towards metavariables. □

Lemma 3.1.2. *For each interim result (σ, Γ), if there exists an equation in Γ that contains an $M \in Var_{MSet}$, then, $\sigma(M) = M$.*

Proof. By induction on the structure of the derivation of the interim result. Clearly, the lemma holds for the initialization $(\mathrm{id}, \Gamma_{ini})$. One sees that there is no rule that adds set variables to Γ. As for σ, E-Set-Application is the only rule that changes it. Assuming Sol does not change set variables occurring in $\{M : \emptyset \stackrel{?}{=} [b_1, ..., b_n]\} \cup \Gamma$, $\{M \mapsto [b_1, ..., b_n]\} \circ Sol$ also does not change set variables occuring in $\Gamma[[b_1, ..., b_n]/M]$, since M was substituted away. □

E-Set-Application: Assume that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{M : \emptyset \stackrel{?}{=} e\} \cup \Gamma$, in particular, $\tau(M) = \tau \circ Sol(M) = \tau \circ Sol(e) = \tau(e)$ (Sol can be removed due to Lemmata 2.2.1 and 3.1.2). Then, $\tau \circ \{M \mapsto e\} \circ Sol = \eta = \tau \circ Sol$ as needed, since on both side of the equation, M, as the only variable the change has an effect, is sent to $\tau(e)$. It remains to show that η also solves $\Gamma[e/M]$, which holds since it solved $\{M : \emptyset \stackrel{?}{=} e\} \cup \Gamma$, in particular Γ already, and $[e/M]$ makes no difference since η, as a solution to $\{M : \emptyset \stackrel{?}{=} e\}$, did not distinguish between M and e anyway.
E-Set-Distribution: Assume $k > 0, m > 0$ and η solves $\{M : [b_1, ..., b_k] \stackrel{?}{=} [b'_1, ..., b'_m]\} \cup \Gamma$, in particular, there exists an injection $\pi : \{1, ..., k\} \to \{1, ..., m\}$ such that $\eta(b_i) = \eta(b'_{\pi(i)})$ for each $i \in \{1, ..., k\}$ and $\eta(M) = [b'_j \mid j \in \{1, ..., m\} \setminus \mathrm{im}(\pi)]$. Then, choosing

$i := \pi(1)$, η also solves $\{b_1 \overset{?}{=} b'_i, M : [b_2, ..., b_k] \overset{?}{=} [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$, since $\eta(b_1) = \eta(b'_{\pi(1)}) = \eta(b'_i)$ and $\eta(M : [b_2, ..., b_k]) = \eta(M) \cup [\eta(b_2), ..., \eta(b_k)] = [b'_j \mid j \in \{1, ..., m\} \setminus \mathrm{im}(\pi)] \cup [b'_j \mid j \in \mathrm{im}(\pi) \setminus [b'_i]] = \eta([b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m])$

E-Set-Clash: Assuming $k > 0$, $M : [b_1, ..., b_k] \overset{?}{=} \emptyset$ can never be solved, making the statement hold trivially.

E-Set-Orientation: Flipping an equation does not change the property of η as a solution.

3.2 Single true set on both sides

3.2.1 Problem statement

In this section (3.2), we examine a variant of the unification problem in which expressions are seen as a true set instead of a multiset.

Definition 3.2.1 (set expressions)**.** *When we write $[b_1, ..., b_n]$ for an expression, we silently assume that all bindings are distinct, i.e. $b_i \neq b_j$ for any $i, j \in \{1, ..., n\}$ such that $i \neq j$. We write $Expr'$ to denote these set expressions.*

Definition 3.2.2 (set-equality of expressions)**.** *Using the notation above, Definition 2.1.2 can be reinterpreted as describing set-equality.*

Definition 3.2.3 (set variables)**.** *We interpret multiset variables as set variables and write $Var_{Set} := Var_{MSet}$.*

Definition 3.2.4 (single set expressions)**.** *An expression of variable bindings with single set variables over the variable space V is defined by:*

$$Expr_V^{SS} ::= Expr'_V \mid V_{Set} : Expr'_V$$

Canonical extension and set substitutions are analogous to multisets (Definitions 3.1.3 and 3.1.4, respectively).

We now lift the one-sidedness restriction introduced in Definition 3.1.5:

Definition 3.2.5. *An element of a unification problem extended by single sets is defined by:*

$$ProbEl^{SS} ::= Expr^{SS} \doteq Expr^{SS}$$

A unification problem $UnifProblem^{SS}$ is a finite set of $ProbEl^{SS}$s.

3.2.2 Solution

*SolverEl*s now use $Expr_M^{SS}$s:

$$SolverEl^{SM} ::= Expr_M^{SS} \overset{?}{=} Expr_M^{SS} \mid Bind_M \overset{?}{=} Bind_M \mid Var_M \overset{?}{=} Var_M$$

Moreover, we introduce a new type of variable:

Definition 3.2.6. *(helper set variables) Elements of the set* $Var_{Help} := \{M_i \overbrace{'...'}^{n} \mid n \in \mathbb{N}_+, M_i \in Var_{MSet}\}$ *are called helper variables.*

Then, the solution to $UnifProblem^{SS}$ is given by the new rules in Figure 3.2 in addition to our former rules from Figures 2.1 and 3.1, reinterpreted as acting over set expressions.

The old rules as well as their proofs can be reused, as they do not rely on the ability of expressions to contain duplicate elements, with one exception in the completeness proof of E-SET-DISTRIBUTION, demanding $\eta(M)$ and $\eta(e)$ to be disjoint if η is a solution and $M : e$ an expression contained in the solver data structure. As such, solutions that do not satisfy this property are disregarded by the new algorithm. For example, the algorithm finds $\{M \mapsto \emptyset\}$ as the only solution to $M : [b_1, ..., b_9] \doteq [b_1, ..., b_9]$, although $\{M \mapsto [b_4, b_2]\}$, $\{M \mapsto [b_8, b_1, b_9]\}$ or $\{M \mapsto [b_8, b_2, b_7, b_6]\}$ etc. would also be solutions. This omission is justified by the fact that, if needed, the missing solutions can be found easily and, apart therefrom, adding them would result in a combinational blow up of the size of the solution set ($2^n - 1$ for each $M : e$ where e is of length n).

We make two further changes in the interpretation of the rules: Firstly, we used to select an element from Γ randomly until now, which can, owing to the last branch of E-BISET-DISTRIBUTION, lead to an infinite loop, e.g.:

$$\begin{array}{rl}
(\quad Sol, & \{M_1 : [b_1] \stackrel{?}{=} M_2 : [b],\ M_2 \stackrel{?}{=} M_1 : [b]\}) \\ \hline
(\{M_2 \mapsto M_2' : [b_1]\} \circ Sol, & \{M_1 \stackrel{?}{=} M_2' : [b],\ M_2' : [b_1] \stackrel{?}{=} M_1 : [b]\}) \\ \hline
(\{M_1 \mapsto M_1' : [b_1]\} \circ Sol, & \{M_1' : [b_1] \stackrel{?}{=} M_2' : [b],\ M_2' \stackrel{?}{=} M_1' : [b]\}) \\ \hline
\multicolumn{2}{c}{\vdots}
\end{array}$$

Both M_1 and M_2 could contain b_1, allowing it to be passed back and forth between them. For the termination of the new algorithm, we demand that, if $|e| < |e'|$, $M : e \stackrel{?}{=} t$ is always preferred over $M' : e' \stackrel{?}{=} t'$. This is not necessary if we restrict set variables to be linear.

Secondly, we view unifiers Sol retrieved by the algorithm as the set of all solutions τ such that $Sol \leq_{[M,Set]} \tau$, i.e. we restrict (cf. Def. 2.1.10) generality of solutions to non-helper variables. For example, $\tau := \{M \mapsto [a = b, c = d, e = f]\}$ is a solution to the problem $\{M : [a = b] \doteq M : [c = d]\}$, which would not be an (unrestricted) instantiation of $Sol := \{M \mapsto M' : [a = b, c = d],\ M' \mapsto M' : [c = d]\}$, which the algorithm retrieves, since, in order to realize $\tau = \lambda \circ Sol$, λ would have to undo $\{M' \mapsto M' : [c = d]\}$, which it cannot. Under restriction to $[M, Set]$, however, $Sol =_{[M,Set]} \{M \mapsto M' : [a = b, c = d]\}$ holds, enabling us to choose $\lambda := \{M' \mapsto [e = f]\}$.

Remark 3.1. During the algorithm in Figure 3.2, for a variable to be fresh it suffices to add an apostrophe to an existing set variable.

Proof. Let (σ, Γ) be an interim result and $M^* \in Var_{Help} \cup Var_{Set}$ such that there exists an equation in Γ that contains M^*. Then, $(M^*)'$ is fresh, since if it was not, it must

E-BISET-TAUTOLOGY
$$\frac{(Sol, \{M : \emptyset \overset{?}{=} M : \emptyset\} \cup \Gamma)}{(Sol, \Gamma)}$$

E-BISET-ORIENTATION
$$\frac{(Sol, \{M_1 : e_1 \overset{?}{=} M_2 : \emptyset\} \cup \Gamma)}{(Sol, \{M_2 : \emptyset \overset{?}{=} M_1 : e_1\} \cup \Gamma)} e_1 \neq \emptyset$$

E-BISET-APPLICATION
$$\frac{(Sol, \{M_1 : \emptyset \overset{?}{=} M_2 : e_2\} \cup \Gamma)}{(\{M_1 \mapsto M_2 : e_2\} \circ Sol, \Gamma[M_2 : e_2/M_1])} M_1 \neq M_2 \text{ or } e_2 \neq \emptyset$$

E-BISET-DISTRIBUTION
$$\frac{(Sol, \{M_1 : [b_1, ..., b_k] \overset{?}{=} M_2 : [b'_1, ..., b'_m]\} \cup \Gamma)}{\begin{array}{c} |_{i=1}^{m} \ (Sol, \{b_1 \overset{?}{=} b'_i, M_1 : [b_2, ..., b_k] \overset{?}{=} M_2 : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma) \\ | \ (\{M_2 \mapsto M'_2 : [b_1]\} \circ Sol, \{M'_1 : [b_2, ..., b_k] \overset{?}{=} M'_2 : [b'_1, ..., b'_m]\} \cup \Gamma[M'_2 : [b_1]/M_2]) \\ \text{where } M'_2 \text{ is fresh and } M'_1 := (\text{if } M_1 = M_2 \text{ then } M'_2 \text{ else } M_1) \end{array}} \begin{array}{l} k>0, \\ m>0 \end{array}$$

Figure 3.2: single set variable on both sides of the equation

have been added by E-BISET-DISTRIBUTION, which eliminates M^* from Γ, disagreeing with the assumption that Γ contains M^*. □

3.2.3 Correctness

Termination

One verifies that any equation of the form $M_1 : e_1 \overset{?}{=} M_2 : e_2$ is treated by a unique rule:

(e_1,e_2)	$M_1 = M_2$	$M_1 \neq M_2$
($\emptyset$,$\emptyset$)	TAUT	APP
($>$,$\emptyset$)	ORIENT	ORIENT
($\emptyset$,$>$)	APP	APP
($>$,$>$)	DIST	DIST

In addition to μ from the simple unification problem as well as $\mu_{RSetVar}$ and μ_{SetVar} reinterpreted from $\mu_{RMSetVar}$ and $\mu_{MSetVar}$ from the one-sidedly extended multiset problem, we monitor $\mu_{minSetExprL}$, the smallest of lengths of set-extended expressions occurring on the left-hand side of equations. Formally:

$$\begin{aligned}\kappa_{minSetExprL} : \mathit{Solver}^{SS} &\longrightarrow \mathbb{N}_0 \cup \{\infty\}\\ t_1 \stackrel{?}{=} t_2 &\longmapsto \begin{cases} |e|, t_1 = M : e\\ \infty, \text{otherwise}\end{cases}\end{aligned}$$

and

$$\begin{aligned}\mu_{minSetExprL} : \mathit{Solver}^{SS} &\longrightarrow \mathbb{N}_0 \cup \{\infty\}\\ \Gamma &\longmapsto \min\{\kappa_{minSetExprL}(\gamma) \mid \gamma \in \Gamma\}\end{aligned}$$

Then, for each rule application, the measure

$$\begin{aligned}\mu^{SS} : \mathit{Solver}^{SM} &\longrightarrow (\mathbb{N}_0 \cup \{\infty\})^8\\ \Gamma &\longmapsto (\mu_{SetVar}(\Gamma), \mu_{RSetVar}(\Gamma), \mu_{minSetExprL}(\Gamma), \mu(\Gamma))\end{aligned}$$

strictly decreases with respect to the lexicographic ordering on $(\mathbb{N}_0 \cup \{\infty\})^8$.

Indeed, no rule from the simple unification problem changes μ_{SetVar}, $\mu_{RSetVar}$ or $\mu_{minSetExprL}$. E-Set-Application decreases μ_{SetVar}. E-Set-Distribution does not change μ_{SetVar} or $\mu_{RSetVar}$ and decreases $\mu_{minSetExprL}$. E-Set-Orientation decreases $\mu_{RSetVar}$ and does not change μ_{SetVar}. E-Biset-Tautology decreases μ_{SetVar}. E-Biset-Orientation might decrease μ (BExpr or ExprE), but only decreases $\mu_{minSetExprL}$ for sure; all other measures stay the same. E-Biset-Application decreases μ_{SetVar}. E-Biset-Distribution decreases $\mu_{minSetExprL}$ for sure and does not increase μ_{SetVar} or $\mu_{RSetVar}$.

Rule	SetVar	RSetVar	minSetExpL	ExprE	BExpr	BindE	V	RMV
S-App	**dec**	dec?	inc?	dec	?	-	-	-
S-Dst	-	-	**dec**	dec?	dec	inc?	-	-
S-Ori	-	**dec**	inc?	dec?	-	-	-	-
Taut	**dec**	dec	inc?	dec	-	-	-	-
Orient	-	-	**dec**	dec?	dec?	-	-	-
App	**dec**	dec	inc?	dec	inc?	-	-	-
Dist1	dec?	dec?	**dec**	dec?	dec	inc	-	-
Dist2	-	dec?	**dec**	-	inc?	-	-	-

Soundness

E-Biset-Tautology: Assume that for any τ that unifies $\{M : \emptyset \stackrel{?}{=} M : \emptyset\} \cup \Gamma$, $\tau \circ \mathit{Sol}$ unifies Γ_{ini}. Let τ' be a unifier for Γ. To be shown is that $\tau' \circ \mathit{Sol}$ unifies Γ_{ini}. Choosing $\tau := \tau'$, τ unifies $\{M : \emptyset \stackrel{?}{=} M : \emptyset\} \cup \Gamma$, since $\{M : \emptyset \stackrel{?}{=} M : \emptyset\}$ is already unified. Hence, $\tau' \circ \mathit{Sol}$ is a unifier for Γ_{ini}.

E-Biset-Orientation: Assume that for any τ that unifies $\{M_1 : e_1 \stackrel{?}{=} M_2 : \emptyset\} \cup \Gamma$,

$\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for Γ'. To be shown is that $\tau' \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau'$, τ unifies $\{M_1 : e_1 \stackrel{?}{=} M_2 : \emptyset\} \cup \Gamma$, since equality is symmetric. Hence, $\tau' \circ Sol$ is a unifier for Γ_{ini}.

E-Biset-Application: Assume that for any τ that unifies $\{M_1 : \emptyset \stackrel{?}{=} M_2 : e_2\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\Gamma[M_2 : e_2/M_1]$. To be shown is that $\tau' \circ \{M_1 \mapsto M_2 : e_2\} \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau' \circ \{M_1 \mapsto M_2 : e_2\}$, τ unifies $\{M_1 : \emptyset \stackrel{?}{=} M_2 : e_2\} \cup \Gamma$, since applying $\{M_1 \mapsto M_2 : e_2\}$ gives $\{M_2 : e_2 \stackrel{?}{=} M_2 : e_2\} \cup \Gamma[M_2 : e_2/M_1]$, which τ' solves, since the former is already solved and the latter is part of the assumption for τ'. Hence, $\tau' \circ \{M_1 \mapsto M_2 : e_2\} \circ Sol$ is a unifier for Γ_{ini}.

E-Biset-Distribution: Assume that for any τ that unifies $\{M_1 : [b_1, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_m]\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}.

Let τ' be a unifier for $\{b_1 \stackrel{?}{=} b'_i, M_1 : [b_2, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$, in particular, $\tau'(b_1) = \tau'(b'_i)$. To be shown is that $\tau' \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau'$, τ unifies $\{M_1 : [b_1, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_m]\} \cup \Gamma$, since both sides of the equation were extended by a binding which are the same under $\tau = \tau'$. Hence, $\tau' \circ Sol$ is a unifier for Γ_{ini}.

Let τ' be a unifier for $\{M'_1 : [b_2, ..., b_k] \stackrel{?}{=} M'_2 : [b'_1, ..., b'_m]\} \cup \Gamma[M'_2 : [b_1]/M_2]$. To be shown is that $\tau' \circ \{M_2 \mapsto M'_2 : [b_1]\} \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau' \circ \{M_2 \mapsto M'_2 : [b_1]\}$, τ unifies $\{M_1 : [b_1, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_m]\} \cup \Gamma$, since applying $\{M_2 \mapsto M'_2 : [b_1]\}$ gives $\{M'_1 : [b_1, ..., b_k] \stackrel{?}{=} M'_2 : [b_1, b'_1, ..., b'_m]\} \cup \Gamma[M'_2 : [b_1]/M_2]$ which τ' solves, since the former is almost the same equation τ' already solved, only that b_1 was added on both sides, which does not affect the validity of τ' as a solution, and the latter is part of the assumption for τ'. Hence, $\tau' \circ \{M_2 \mapsto M'_2 : [b_1]\} \circ Sol$ is a unifier for Γ_{ini}.

Completeness

Lemma 3.2.1. *For each interim result (σ, Γ), if there exists an equation in Γ that has an $M : e \in Expr_M^{SM}$ on one (or both) of its sides, then, $\sigma(M : e) = M : e$.*

Proof. By induction on the derivation of the interim result. The statement clearly holds for the initialization. E-Biset-Distribution can add set variables, which are, however, fresh, such that σ does not affect them. There is no other rule adds set variables.

E-Set-Application and E-Biset-Distribution both make σ change an additional set variable, which is, however, eliminated immediately from Γ.

In E-Biset-Application, if $M_1 \neq M_2$, M_1 is eliminated from Γ, making the change $\{M_1 \mapsto M_2 : e_2\}$ not affect this lemma. If $M_1 = M_2$, σ is extended by $\{M_2 \mapsto M_2 : e_2\}$ and Γ is changed to $\Gamma[M_2 : e_2/M_2]$, i.e. any occurence of $M_2 : e$ in Γ is replaced by $M_2 : (e \cup e_2)$. Then, applying $\{M_2 \mapsto M_2 : e_2\} \circ \sigma$ still complies with our statement, since, due to Lemma 2.2.1 ($\sigma(e) = e$) and the induction hypothesis ($\sigma(M_2 : e_2) = M_2 : e_2$),

$$\{M_2 \mapsto M_2 : e_2\} \circ \sigma(M_2 : (e \cup e_2)) = \{M_2 \mapsto M_2 : e_2\}(M_2 : (e \cup e_2)) = M_2 : (e \cup e_2).$$

All other rules leave σ unchanged. □

E-Biset-Tautology: If η solves $\{M : \emptyset \stackrel{?}{=} M : \emptyset\} \cup \Gamma$, it also must have solved its subset Γ.
E-Biset-Orientation: Flipping an equation does not change the property of η as a solution.
E-Biset-Application: Assume $M_1 \neq M_2$ or $e_2 \neq \emptyset$ and that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{M_1 : \emptyset \stackrel{?}{=} M_2 : e_2\} \cup \Gamma$, in particular, $e := \eta(M_1) = \eta(M_2 : e_2)$. From Lemma 3.2.1 we know that one can write $Sol(M_1) = M_1 : e_1'$. It is clear that $\tau(e_1') \subseteq \tau(e_1') \cup \tau(M_1) = \tau(M_1 : e_1') = e$. Then, $\tau \circ Sol = \eta = \tau \circ \{M_1 \mapsto Sol(M_2 : e_2)\} \circ Sol$ as needed, since on both sides of the equation, for M_1 — the only variable the change has an effect on — the following holds:

$$\begin{aligned}
\tau \circ \{M_1 \mapsto M_2 : e_2\} \circ Sol(M_1) =& \tau \circ \{M_1 \mapsto M_2 : e_2\}(M_1 : e_1') \\
=& \tau(M_2 : (e_1' \cup e_2)) \\
=& \tau(M_2 : e_2) \cup \tau(e_1') \\
=& \tau(M_2 : e_2) \stackrel{3.2.1}{=} \tau(Sol(M_2 : e_2)) \qquad = \tau \circ Sol(M_1)
\end{aligned}$$

It remains to show that η also solves $\Gamma[M_2 : e_2/M_1]$, which holds since it already solved Γ and $[M_2 : e_2/M_1]$ makes no difference since η did not distinguish between $M_2 : e_2$ and M_1 anyway.
E-Biset-Distribution: Assume $k > 0, m > 0$ and that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{M_1 : [b_1, ..., b_k] \stackrel{?}{=} M_2 : [b_1', ..., b_m']\} \cup \Gamma$, in particular, $\eta(M_1 : [b_1, ..., b_k]) = \eta(M_2 : [b_1', ..., b_m'])$. There are two cases to consider: In case $\eta(b_1) = \eta(b_i')$ for an i, then, η also solves $\{b_1 \stackrel{?}{=} b_i', M_1 : [b_2, ..., b_k] \stackrel{?}{=} M_2 : [b_1', ..., b_{i-1}', b_{i+1}', ..., b_m']\} \cup \Gamma$ by the same argumentation as in the proof for E-Set-Distribution.
In case $\tau \circ Sol(b_1) = \tau(b_1) \in \eta(M_2) \setminus \eta([b_1', ..., b_m'])$ (Sol can be removed due to Lemma 2.2.1), choosing $\tau' := \tau \circ \{M_2' \mapsto \eta(M_2) \setminus [\tau(b_1)]\}$,

$$\begin{aligned}
\tau' \circ \{M_2 \mapsto M_2' : [b_1]\} \circ Sol(M_2) =& \tau' \circ \{M_2 \mapsto M_2' : [b_1]\}(M_2 : e_2') \\
=& \tau'(M_2' : ([b_1] \cup e_2')) \\
=& \tau'(M_2' : e_2') \cup \tau'([b_1]) \\
=& (\tau(M_2 : e_2') \setminus [\tau(b_1)]) \cup [\tau(b_1)] \\
=& \tau(M_2 : e_2') = \eta(M_2) = \qquad \tau \circ Sol(M_2),
\end{aligned}$$

showing $\tau' \circ \{M_2 \mapsto M_2' : [b_1]\} \circ Sol =_{[M,Set]} \tau \circ Sol$.

It remains to show that η also solves $\{M_1' : [b_2, ..., b_k] \stackrel{?}{=} M_2' : [b_1', ..., b_m']\} \cup \Gamma[M_2' : [b_1]/M_2]$. Since all properties to be shown are restricted to non-helper variables, we can make the following transformation: $\eta \geq \{M_2 \mapsto \eta(M_2)\} =_{[Var_{MSet}]} \{M_2' \mapsto \eta(M_2) \setminus \eta(b_1)\} \circ \{M_2 \mapsto M_2' : \eta([b_1])\} = \{M_2 \mapsto \eta(M_2), M_2' \mapsto \eta(M_2) \setminus \eta(b_1)\}$, in other words, $\eta(M_2) = \eta([b_1]) \cup \eta(M_2')$. Then, one sees immediately that $[M_2' : [b_1]/M_2]$ makes no difference and, if $M_1 = M_2$, $\eta(M_2' : [b_2, ..., b_k]) = \eta(M_1 : [b_1, ..., b_k]) \setminus \eta([b_1]) = \eta(M_2 : [b_1', ..., b_m']) \setminus \eta([b_1]) = \eta(M_2' : [b_1', ..., b_m'])$ as well as, if $M_1 \neq M_2$, $\eta(M_1 : [b_2, ..., b_k]) = \eta(M_1 : [b_1, ..., b_k]) \setminus \eta([b_1]) = \eta(M_2 : [b_1', ..., b_m']) \setminus \eta([b_1]) = \eta(M_2' : [b_1', ..., b_m'])$.

3.3 Single multiset on both sides

3.3.1 Problem statement

In this section (3.3), expressions are seen as a multiset again, instead of a true set like in Section 3.2. As in Definitions 3.2.4 and 3.2.5, we allow at most one multiset variable in both sides of the equation.

3.3.2 Solution

All constructs and reinterpretations introduced in Section 3.2.2 remain valid. Concretely:

1. We use helper variables (Def. 3.2.6).

2. We do not choose rules randomly, as this could result in an infinite loop, due to the second branch of E-MSET-DISTRIBUTION. Instead, whenever $|e| < |e'|$, we favor $M : e \stackrel{?}{=} t$ over $M' : e' \stackrel{?}{=} t'$.

3. $\leq$ is restricted to non-helper variables.

Under these prerequisites, the new rules in Figure 3.3, together with the rules in Figure 3.1 and 2.1, solve the unification problem with single multisets on both sides.

3.3.3 Correctness

Termination

One verifies that any equation of the form $M_1 : e_1 \stackrel{?}{=} M_2 : e_2$ is treated by a unique rule:

(e_1,e_2)	$M_1 = M_2$	$M_1 \neq M_2$
$(\emptyset,\emptyset)$	TAUT	APP
$(>,\emptyset)$	ORIENT	ORIENT
$(\emptyset,>)$	CLASH	APP
$(>,>)$	SEM-TAUT	DISTR

We use the same measure as in the version for true sets, where μ_{SetVar} and $\mu_{RSetVar}$ are used again (as in the one-sided multiset extension), and $\mu_{minSetExprL}$ is reinterpreted to act on multisets under the name $\mu_{minMSExprL}$.

E-MSET-CLASH terminates immediately. E-MSET-SEMI-TAUTOLOGY reduces μ_{SetVar}. All other rules are either the same (Figures 2.1 and 3.1), only a renaming (TAUTOLOGY, ORIENTATION) or a subset of a rule from Figure 3.2 (APPLICATION, DISTRIBUTION).

E-Mset-Tautology
$$\frac{(Sol, \{M : \emptyset \stackrel{?}{=} M : \emptyset\} \cup \Gamma)}{(Sol, \Gamma)}$$

E-Mset-Orientation
$$\frac{(Sol, \{M_1 : e_1 \stackrel{?}{=} M_2 : \emptyset\} \cup \Gamma)}{(Sol, \{M_2 : \emptyset \stackrel{?}{=} M_1 : e_1\} \cup \Gamma)} e_1 \neq \emptyset$$

E-Mset-Application
$$\frac{(Sol, \{M_1 : \emptyset \stackrel{?}{=} M_2 : e_2\} \cup \Gamma)}{(\{M_1 \mapsto M_2 : e_2\} \circ Sol, \Gamma[M_2 : e_2/M_1])} M_1 \neq M_2$$

E-Mset-Clash
$$\frac{(Sol, \{M : \emptyset \stackrel{?}{=} M : e\} \cup \Gamma)}{Fail} e \neq \emptyset$$

E-Mset-Semi-Tautology
$$\frac{(Sol, \{M : e_1 \stackrel{?}{=} M : e_2\} \cup \Gamma)}{(Sol, \{e_1 \stackrel{?}{=} e_2\} \cup \Gamma)} e_1 \neq \emptyset, e_2 \neq \emptyset$$

E-Mset-Distribution
$$\frac{(Sol, \{M_1 : [b_1, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_m]\} \cup \Gamma)}{\begin{array}{c} |_{i=1}^{m} \ (Sol, \{b_1 \stackrel{?}{=} b'_i, M_1 : [b_2, ..., b_k] \stackrel{?}{=} M_2 : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma) \\ | \ (\{M_2 \mapsto M'_2 : [b_1]\} \circ Sol, \{M_1 : [b_2, ..., b_k] \stackrel{?}{=} M'_2 : [b'_1, ..., b'_m]\} \cup \Gamma[M'_2 : [b_1]/M_2]) \\ \text{where } M'_2 \text{ is fresh} \end{array}} {\scriptstyle M_1 \neq M_2, \atop k>0, m>0}$$

Figure 3.3: single set variable on both sides of the equation

Soundness

E-MSET-CLASH: There is nothing to show.
E-MSET-SEMI-TAUTOLOGY: Assume that for any τ that unifies $\{M : e_1 \stackrel{?}{=} M : e_2\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\{e_1 \stackrel{?}{=} e_2\} \cup \Gamma$, in particular, $\tau'(e_1) = \tau'(e_2)$. Then, τ' unifies $\{M : e_1 \stackrel{?}{=} M : e_2\} \cup \Gamma$, since $\tau'(M : e_1) = \tau'(M) \cup \tau'(e_1) = \tau'(M) \cup \tau'(e_2) = \tau'(M : e_2)$ and τ' already unifies Γ. Thus, applying the assumption, $\tau' \circ Sol$ unifies Γ_{ini}, as needed.
All other rules can be proven by using their BISET-equivalents.

Completeness

E-MSET-CLASH: Assuming $e \neq \emptyset$, $\{M : \emptyset \stackrel{?}{=} M : e\}$ can never be unified, as $\eta(\emptyset) = \emptyset \neq \eta(e)$ in contradiction to what $\eta(M : \emptyset) = \eta(M : e)$ would imply with Lemma 2.1.1, confirming the statement trivially.
E-MSET-SEMI-TAUTOLOGY: Assume η unifies $\{M : e_1 \stackrel{?}{=} M : e_2\} \cup \Gamma$, in particular, $\eta(M : e_1) = \eta(M) \cup \eta(e_1) = \eta(M) \cup \eta(e_2) = \eta(M : e_2)$, implying $\eta(e_1) = \eta(e_2)$ with Lemma 2.1.1, precisely what is needed for η to unify $\{e_1 \stackrel{?}{=} e_2\} \cup \Gamma$.
All other rules can be proven by using their BISET-equivalents.

3.4 Full multiset extension

3.4.1 Problem statement

We now lift the restriction that only one set variable per expression is allowed.

Definition 3.4.1 (fully multiset-extended expressions)**.** *A fully multiset-extended expression of variable bindings over the variable space V is defined by:*

$$Expr_V^F ::= Expr_V \mid \mathfrak{M} : Expr_V$$

where $\mathfrak{M} = [M_i, ..., M_j]$ is a multiset over Var_{MSet}. We write $[M^n]$ for $[\underbrace{M, ..., M}_{n}]$ and omit the square braces when $n = 1$ and inside an expression.

Definition 3.4.2. *An element of a fully multiset-extended unification problem is defined by:*

$$ProbEl^F ::= Expr^F \doteq Expr^F$$

A fully multiset-extended unification problem is a finite set of $ProbEl^F$s.

Canonical extensions and multiset substitutions are defined analogous to Definitions 3.1.3 and 3.1.4.

3.4.2 Solution

We introduce new types of helper variables:

Definition 3.4.3 (branching helper variables)**.** *Let $M, N \in Var_{MSet}$ be variables, $n \in \mathbb{N}_0$. Then, $\mathcal{T}(M,N)\underbrace{'..'}_{n}$ is a branching helper variable.*

Definition 3.4.4 (memorizing helper variables)**.** *Let $n \in \mathbb{N}_0$. A memorizing helper variable $\mathcal{M}_\alpha\underbrace{'..'}_{n}$ carries an expression α as part of its defining data.*

Definition 3.4.5 (block helper variables)**.** *Let $i, n \in \mathbb{N}_0$. A meta-block helper variable $\mathbf{N}_i\underbrace{'..'}_{n}$ distinguishes itself by sometimes (but not necessarily) being written in bold-face. A ground-block helper variable $\mathcal{N}_i\underbrace{'..'}_{n}$ is written calligraphic font.*

Definition 3.4.6 (block equation)**.** *We call equations of the form $[\mathcal{M}^c] : e_1 \stackrel{?}{=} \mathbf{N} : e_2$ "block".*

All constructs and reinterpretations introduced in Section 3.2.2 remain valid (the use of normal helper variables, order of rule applications, restriction of $\leq$). Furthermore, the rule X-Semi-Tautology is favoured over all other rules, overriding the preference of $M : e \stackrel{?}{=} t$ over $M' : e' \stackrel{?}{=} t'$ for $|e| < |e'|$, if necessary. Also, block equations are preferred over other equations.

The solution is obtained by the rules in Figure 3.4 in addition to Figure 2.1 excluding the E-rules, where, in X-Partition, $Z(e, r)$ is the set of "r-partitions" of e defined as follows:

Definition 3.4.7 (r-partitions)**.** *Let $r \in \mathbb{N}_+$. The set of r-partitions of the simple expression ε is defined as*

$$Z(\varepsilon, r) := \{\zeta : \{1..r\} \to 2^\varepsilon \mid \bigcup_{j \in \{1..m\}} \zeta(j) = \varepsilon\}.$$

3.4.3 Correctness

Termination

One verifies that any non-block equation of the form $\mathfrak{M}_1 : e_1 \stackrel{?}{=} \mathfrak{M}_2 : e_2$ as well as any block equation is treated by a unique rule: if $\mathfrak{M}_1 \cap \mathfrak{M}_2 \neq \emptyset$, X-Semi-Tautology is chosen and else as in Table 3.1. As soon as block equations are introduced by X-Partition into the solver set, they are preferred over the other equations until every block equation is eliminated by X-Rep-Base. During that process, all these block equations can be treated independently from each other, as no two block equations

X-Semi-Tautology

$$\frac{(Sol, \{\mathfrak{M} : e_1 \stackrel{?}{=} \mathfrak{N} : e_2\} \cup \Gamma)}{(Sol, \{(\mathfrak{M} \setminus \mathfrak{N}) : e_1 \stackrel{?}{=} (\mathfrak{N} \setminus \mathfrak{M}) : e_2\} \cup \Gamma)} \mathfrak{M} \cap \mathfrak{N} \neq \emptyset$$

If $\mathfrak{M} \cap \mathfrak{N} = \emptyset$:

X-Orientation

$$\frac{(Sol, \{\mathfrak{M}_1 : e_1 \stackrel{?}{=} \mathfrak{M}_2 : \emptyset\} \cup \Gamma)}{(Sol, \{\mathfrak{M}_2 : \emptyset \stackrel{?}{=} \mathfrak{M}_1 : e_1\} \cup \Gamma)} {}^{e_1 \neq \emptyset}_{\text{eqn. not block}}$$

X-Clash

$$\frac{(Sol, \emptyset \stackrel{?}{=} \mathfrak{M} : e)}{\mathit{Fail}} e \neq \emptyset$$

X-Distribution

$$\frac{(Sol, \{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} [N_1, ..., N_p] : [b'_1, ..., b'_m]\} \cup \Gamma)}{\begin{array}{c} |_{i=1}^{m} \ (Sol, \{b_1 \stackrel{?}{=} b'_i, \mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} [N_1, ..., N_p] : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma) \\ |_{i=1}^{p} \ (\nu \circ Sol, \{\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} (\nu([N_1, ..., N'_i, ..., N_p])) : [b'_1, ..., b'_m]\} \cup \nu(\Gamma)) \\ \text{where } N'_i \text{ is fresh and } \nu = \{N_i \mapsto N'_i : [b_1]\} \end{array}} {}^{k>0,}_{m>0}$$

X-Emp-Application

$$\frac{(Sol, \{[M_1, ..., M_q] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : \emptyset\} \cup \Gamma)}{(\mu\nu \circ Sol, \mu\nu(\Gamma))} \text{eqn. is not block}$$

$$\text{where } \mu = \{M_i \mapsto [\mathcal{T}_{(M_i, N_j)} \mid j \in [1..p]] \mid i \in \{1..q\}\}$$
$$\text{and } \nu = \{N_j \mapsto [\mathcal{T}_{(M_i, N_j)} \mid i \in [1..q]] \mid j \in \{1..p\}\}$$

X-Partition

$$\frac{(Sol, \{[M_1, M_1, ..., M_r, M_r] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : e\} \cup \Gamma)}{|_{\zeta \in Z(e,r)} \ (\mu \circ Sol, \{[\mathcal{M}_{\alpha_i}^{c_i}] \stackrel{?}{=} \mathbf{N}_i : \zeta(i) \ |_{i \in \{1..r\}}\} \cup \{[\mathcal{N}_1 .. \mathcal{N}_p] \stackrel{?}{=} [\mathbf{N}_1 .. \mathbf{N}_r]\} \cup \mu(\Gamma))} {}^{r>0,}_{\substack{e \neq \emptyset, \\ \text{eqn n-blc}}}$$

$$\text{where } M_i \text{ occurs } c_i \text{ times in } [M_1, M_1, ..., M_r, M_r],$$
$$\mu = \{M_i \mapsto \mathcal{M}_{\alpha_i} \ |_{i \in \{1..r\}}\} \circ \{N_j \mapsto [\mathcal{N}_j] \cup [\mathcal{T}_{(M_i, N_j)} \ |_{i \in [1,1..r]}] \ |_{j \in \{1..p\}}\}$$
$$\text{and } \alpha_i = [\mathcal{T}_{(M_i, N_j)} \mid j \in \{1..p\}]$$

X-Rep-Distribution

$$\frac{(Sol, \{[\mathcal{M}^c] : \emptyset \stackrel{?}{=} \mathbf{N} : [b_1, ..., b_n]\} \cup \Gamma)}{(\{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ Sol, \{[(\mathcal{M}')^c] : [b_1^{c-1}] \stackrel{?}{=} \mathbf{N} : [b_2, ..., b_n]\} \cup \Gamma[\mathcal{M}' : [b_1]/\mathcal{M}])} n > 0$$

X-Rep-Base

$$\frac{(Sol, \{[\mathcal{M}_\alpha^c] : e \stackrel{?}{=} \mathbf{N} : \emptyset\} \cup \Gamma)}{(\{\mathcal{M}_\alpha \mapsto \alpha, \ \mathbf{N} \mapsto e\} \circ Sol, \Gamma[\alpha/\mathcal{M}_\alpha, \ e/\mathbf{N}])}$$

Figure 3.4: Algorithm for the unification problem fully extended by multisets

(e_1,e_2)	$\mathfrak{M}_1 = \emptyset$	$\mathfrak{M}_1 \neq \emptyset$	block
$(\emptyset,\emptyset)$	X-Emp-Application		X-Rep-Base
$(>,\emptyset)$	X-Orientation		
$(\emptyset,>)$	X-Clash	X-Partition	X-Rep-Distribution
$(>,>)$	X-Distribution		

Table 3.1: Rule switch

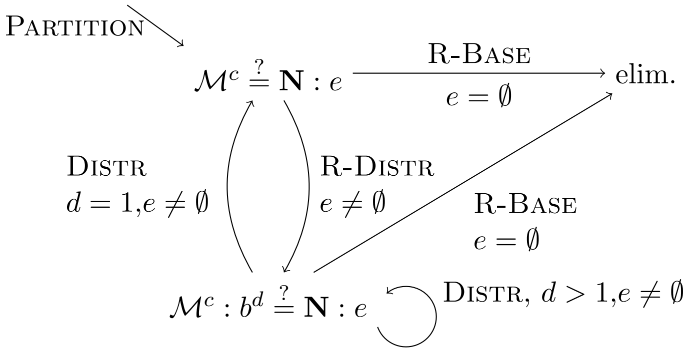

Figure 3.5: Treatment of block equations

share the same set variable. This process, visualized in Figure 3.5, terminates, since the measures decrease as shown in Table 3.2. As can be seen in the visualization, all block equations end in X-Rep-Base, such that all $\mathbf{N}_i$ created in X-Partition will be instantiated by some expression e_i. Thus, X-Partition, X-Rep-Distribution and X-Rep-Base can be, for the sake of this proof, shortened to a single rule, X-S-Partition, as shown in Figure 3.6. Then, the measures decrease as shown in Table 3.3.

Soundness

X-Semi-Tautology: Assume $\mathfrak{M} \cap \mathfrak{N} \neq \emptyset$ and that for any τ that unifies $\{\mathfrak{M} : e_1 \stackrel{?}{=} \mathfrak{N} : e_2\} \cup \Gamma$, $\tau \circ \mathit{Sol}$ unifies Γ_{ini}. Let τ' be a unifier for $\{(\mathfrak{M} \setminus \mathfrak{N}) : e_1 \stackrel{?}{=} (\mathfrak{N} \setminus \mathfrak{M}) : e_2\} \cup \Gamma$.

X-S-Partition

$$\frac{(\mathit{Sol}, \{[M_1, M_1, ..., M_r, M_r] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : e\} \cup \Gamma)}{|_{\text{many branches}}\ (\mu \circ \mathit{Sol}, \{[\mathcal{N}_1, ..., \mathcal{N}_p] \stackrel{?}{=} \bigcup_{i \in \{1..r\}} e_i\} \cup \mu(\Gamma))} \begin{array}{l} r{>}0, \\ e{\neq}\emptyset, \\ \text{eqn n-blc} \end{array}$$

where μ treats the M_i and N_j appropriately.

Figure 3.6: Shortened partition

Rule	# of block eqn	$\|e\|$	d
Distr	-	dec?	**dec**
Rep-Dst	-	**dec**	inc?
Rep-Base	**dec**	-	dec?

Table 3.2: Changes in the measures (block)

Rule	ExprE	minBEL	SetVar	BExpr	BindE	VarE	RMVar
V-Taut	-	-	-	-	-	**dec**	dec?
V-App	dec?	-	-	dec?	dec?	**dec**	dec?
V-Ori	-	-	-	-	-	dec?	**dec**
B-Dec	-	-	-	-	**dec**	inc?	inc?
E-Taut	**dec**	inc?	-	-	-	-	-
X-SemT	-	-	**dec**	-	-	-	-
X-Ori	-	**dec**	dec?	-	-	-	-
X-Dst1	dec?	**dec**	?	?	-	-	-
X-Dst2	dec?	**dec**	?	?	-	-	-
X-E-App	**dec**	inc?	?	-	-	-	-
X-S-Part	dec?	-	**dec**	inc?	-	-	-

Table 3.3: Changes in the measures

Then $\tau := \tau'$ unifies $\{\mathfrak{M} : e_1 \stackrel{?}{=} \mathfrak{N} : e_2\} \cup \Gamma$ since Γ was not changed and both sides of the equation were extended by the same set of set variables, namely $\mathfrak{M} \cap \mathfrak{N}$. Hence, $\tau' \circ Sol$ unifies Γ_{ini}.
X-Orientation: Clear.
X-Clash: There is nothing to show.
X-Distribution: Assume that for any τ that unifies Γ, $\tau \circ Sol$ unifies Γ_{ini}.
Let τ' be a unifier for $\{b_1 \stackrel{?}{=} b'_i, \mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$. Then $\tau := \tau'$ unifies $\{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_m]\} \cup \Gamma$ since the latter was already unified and the former is almost $\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]$ which τ also unified already, only that b_1 and b'_1 were added to each side respectively, which are not distinguished under τ. Hence $\tau' \circ Sol$ unifies Γ_{ini}.
Now, let τ' be a unifier for $\{\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} (\mathfrak{N}[N'_i : [b_1]/N_i] \setminus [b_1]) : [b'_1, ..., b'_m]\} \cup \Gamma[N'_i : [b_1]/N_i]$. We point out $\mathfrak{M} \cap \mathfrak{N} = \emptyset$, since potential intersections were already removed by X-Semi-Tautology. Thus, we can safely assume $\{N \mapsto t\}\mathfrak{M} = \mathfrak{M}$ for any $N \in \mathfrak{N}$. Then $\tau := \tau' \circ \{N_i \mapsto N'_i : [b_1]\}$ unifies $\{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_m]\} \cup \Gamma$ since applying $\{N_i \mapsto N'_i : [b_1]\}$ yields $\{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} \mathfrak{N}[N'_i : [b_1]/N_i] : [b'_1, ..., b'_m]\} \cup \Gamma[N'_i : [b_1]/N_i]$, which τ' unifies, since the latter is part of the assumption and the former is $\{\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} (\mathfrak{N}[N'_i : [b_1]/N_i] \setminus [b_1]) : [b'_1, ..., b'_m]\}$ with just a b_1 added at both sides. Hence $\tau' \circ \{N_i \mapsto N'_i : [b_1]\} \circ Sol$ unifies Γ_{ini}.

X-Emp-Application: Assume the equation is non-block and that for any τ that unifies $\{[M_1,...,M_q] : \emptyset \stackrel{?}{=} [N_1,...,N_p] : \emptyset\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\mu\nu\Gamma$. To be shown is that $\tau'\mu\nu \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau'\mu\nu$, τ unifies $\{[M_1,...,M_q] : \emptyset \stackrel{?}{=} [N_1,...,N_p] : \emptyset\} \cup \Gamma$, since applying $\mu\nu$ yields $\{[\mathcal{T}_{(M_i,N_j)} \mid (i,j) \in \{1..q\} \times \{1..p\}] : \emptyset \stackrel{?}{=} [\mathcal{T}_{(M_i,N_j)} \mid (i,j) \in \{1..q\} \times \{1..p\}] : \emptyset\} \cup \mu\nu(\Gamma)$, which τ' unifies, since the latter is part of the assumption and the former is already unified. Hence, $\tau'\mu\nu \circ Sol$ unifies Γ_{ini}.
X-Partition: For $[\mathcal{T}_{(M_i,N_j)} \mid i \in [1..r], j \in [1..p]]$ we write $[\mathcal{T}_{M,N} \mid_{ij}]$. This rule is *not* sound in itself, but only in combination with X-Rep-Base. Assume the equation is non-block, $r > 0$, $e \neq \emptyset$ and that for any τ that unifies $\{[M_1, M_1, ..., M_r, M_r] : \emptyset \stackrel{?}{=} [N_1,...,N_p] : e\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let ζ be an r-partition of e and τ' a unifier for $\{[\mathcal{M}_{\alpha_i}^{c_i}] \stackrel{?}{=} \mathbf{N}_i : \zeta(i) \mid_{i\in\{1..r\}}\} \cup \{[\mathcal{N}_1..\mathcal{N}_p] \stackrel{?}{=} [\mathbf{N}_1..\mathbf{N}_r]\} \cup \mu(\Gamma)$. To be shown is that $\tau'\mu \circ Sol$ unifies Γ_{ini}: Choosing $\tau := \tau'\mu$, τ unifies Γ. That τ unifies Γ (τ' unifies $\mu(\Gamma)$) is part of the assumption. Applying μ on the input equation yields $\bigcup_{i\in\{1..r\}}[\mathcal{M}_{\alpha_i}^{c_i}] : \emptyset \stackrel{?}{=} [\mathcal{N}_1,...,\mathcal{N}_p] \cup [\mathcal{T}_{M,N} \mid_{ij}] : e$. Since τ' unifies $[\mathcal{N}_1..\mathcal{N}_p] \stackrel{?}{=} [\mathbf{N}_1..\mathbf{N}_r]$, we can change the right-hand side to $[\mathbf{N}_1,...,\mathbf{N}_r] \cup [\mathcal{T}_{M,N} \mid_{ij}] : e$. Similarly, using the fact that τ' unifies $[\mathcal{M}_{\alpha_i}^{c_i}] \stackrel{?}{=} \mathbf{N}_i : \zeta(i)$ for each i, we can turn the left-hand side into $[\mathbf{N}_1,...,\mathbf{N}_r] : e$. The new equation, $[\mathbf{N}_1,...,\mathbf{N}_r] : e \stackrel{?}{=} [\mathbf{N}_1,...,\mathbf{N}_r] \cup [\mathcal{T}_{M,N} \mid_{ij}] : e$, is not unified as the left-hand side is missing a $[\mathcal{T}_{M,N} \mid_{ij}]$. We augment this later in X-Rep-Base, by adding an extra $\alpha_i = [\mathcal{T}_{(M_i,N_j)} \mid j \in \{1..p\}]$ to each $\mathbf{N}_i$, only in the first equation. In other words, provided that τ' is an "almost-unifier" in the sense that it is an (in a coordinated manner) ill-defined substitution that unifies the $\mathbf{N}_i$ in the first equation against too much, $\tau'\mu \circ Sol$ is indeed a unifier for Γ_{ini}.
X-Rep-Distribution: Assume that for any τ that unifies $\{[\mathcal{M}^c] : \emptyset \stackrel{?}{=} \mathbf{N} : [b_1,...,b_n]\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\{[(\mathcal{M}')^c] : [b_1^{c-1}] \stackrel{?}{=} \mathbf{N} : [b_2,...,b_n]\} \cup \Gamma[\mathcal{M}' : [b_1]/\mathcal{M}]$. To be shown is that $\tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\}$, τ unifies $\{[\mathcal{M}^c] : \emptyset \stackrel{?}{=} \mathbf{N} : [b_1,...,b_n]\} \cup \Gamma$, since applying $\{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\}$ yields $[(\mathcal{M}')^c] : [b_1^c] \stackrel{?}{=} \mathbf{N} : [b_1,...,b_n]$, which is almost exactly the output equation, with the only difference that a b_1 was added to both sides. This is, by assumption, unified by τ'. Hence, $\tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ Sol$ is a unifier for Γ_{ini}.
X-Rep-Base: This rule is, again, not sound in itself, but only in combination with X-Partition. Assume that for any τ that unifies $\{[\mathcal{M}_\alpha^c] : e \stackrel{?}{=} \mathbf{N} : \emptyset\} \cup \Gamma$, $\tau \circ Sol$ unifies Γ_{ini}. Let τ' be a unifier for $\Gamma[\alpha/\mathcal{M}_\alpha,\ e/\mathbf{N}]$. To be shown is that $\tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ Sol$ unifies Γ_{ini}. Choosing $\tau := \tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\}$, τ unifies $\{[\mathcal{M}_\alpha^c] : e \stackrel{?}{=} \mathbf{N} : \emptyset\} \cup \Gamma$, since applying $\{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\}$ yields $\{[\alpha^c] : e \stackrel{?}{=} e\} \cup \Gamma[\alpha/\mathcal{M}_\alpha,\ e/\mathbf{N}]$. The unification of the latter is part of the assumption for τ'. The former equation is not unified, as the left-hand side has an extra $[\alpha^c]$. But this was exactly what was needed to augment the missing variables in X-Partition. Hence, provided that Sol was missing exactly $[\alpha^c]$ for $\mathbf{N}$ (which it does), $\tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ Sol$ unifies Γ_{ini} indeed.

Completeness

Lemma 3.4.1. *If (σ, Γ) is an interim result and M_i, N_j are multiset variables occuring in Γ, $\mathcal{T}_{(M_i,N_j)}$ does not occur in Γ. $\mathcal{M}_{\alpha_i}$ (where α is as described in* X-Partition*) and $\mathbf{N}_j$ as well as $\mathcal{N}_j$ also do not occur in Γ*

Proof. X-Emp-Application and X-Partition introduce branching helper variables, but eliminate all of the affected variables immediately. The same is the case for the remaining types of helper variables. □

Lemma 3.4.2. *If (σ, Γ) is an interim result and Γ contains $M \in \mathrm{Var}_{MSet}$, then $\sigma(M) = M$.*

Proof. X-Distribution, X-Emp-Application and X-Partition add set variables, but only fresh ones (Lemma 3.4.2), not affecting σ. Other rules do not add variables. X-Distribution, X-Emp-Application, X-Partition and X-Rep-Base also change σ to touch additional set variables, of which the affected ones, however, are eliminated immediately. □

X-Semi-Tautology: Assume η solves $\{\mathfrak{M} : e_1 \stackrel{?}{=} \mathfrak{N} : e_2\} \cup \Gamma$. Then, η also solves $\{(\mathfrak{M} \setminus \mathfrak{N}) : e_1 \stackrel{?}{=} (\mathfrak{N} \setminus \mathfrak{M}) : e_2\} \cup \Gamma$, as the same set of set variables were removed from both sides of the equation, namely $\mathfrak{M} \cap \mathfrak{N}$.
X-Orientation: Clear.
X-Clash: Holds trivially, since $\emptyset \stackrel{?}{=} \mathfrak{M} : e$ can never be under the assumption $e \neq \emptyset$.
X-Distribution: Assume that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_m]\} \cup \Gamma$.
For b_1, there are two cases to consider. Firstly, if there exists an $i \in \{1, ..., m\}$ such that $\eta(b_1) = \eta(b'_i)$, then, η also solves $\{b_1 \stackrel{?}{=} b'_i, \mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma$, since it solved Γ already, solving $b_1 \stackrel{?}{=} b'_i$ was the assumption for this case and $\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\}$ is just $\{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} \mathfrak{N} : [b'_1, ..., b'_m]\}$ with a binding removed from each side which are the same under η.
Secondly, if there exists an $i \in \{1, ..., p\}$ such that $\eta(b_1) \in \eta(N_i)$, then, choosing $\tau' := \tau \circ \{N'_i \mapsto \eta(N_i) \setminus [\eta(b_1)]\}$, for N_i, the only variable the change could have an effect,

$$
\begin{aligned}
\tau' \circ \{N_i \mapsto N'_i : [b_1]\} \circ Sol(N_i) &= \tau' \circ \{N_i \mapsto N'_i : [b_1]\}(N_i) \\
&= \tau'(N'_i : [b_1]) \\
&= \tau(\eta(N_i) \setminus [b_1] : [b_1]) \\
&= \tau(\eta(N_i)) = \eta(N_i)
\end{aligned}
$$

and hence $\tau' \circ \{N_i \mapsto N'_i : [b_1]\} \circ Sol(N_i) = \eta$ as needed. It remains to show that η also solves $\{\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} (\mathfrak{N}[N'_i : [b_1]/N_i] \setminus [b_1]) : [b'_1, ..., b'_m]\} \cup \Gamma[N'_i : [b_a]/N_i]$. Under restriction to non-helper variables, one can make the following transformation: $\eta \geq \{N_i \mapsto \eta(N_i)\} =_{[\neg Var_{Help}]} \{N'_i \mapsto \eta(N_i) \setminus \eta([b_1])\} \circ \{N_i \mapsto N'_i : \eta([b_1])\} = \{N_i \mapsto \eta(N_i), N'_i \mapsto \eta(N_i) \setminus \eta([b_1])\}$, in other words, $\eta(N_i) = \eta([b_1]) \cup \eta(N'_i)$. Then, one sees that $[N_i' : [b_1]/N_i]$ makes no difference on Γ and $\eta(\mathfrak{M} : [b_2, ..., b_k]) = \eta(\mathfrak{M} : [b_1, ..., b_k]) \setminus \eta([b_1]) = \eta(\mathfrak{N} : [b_1', ..., b_m']) \setminus \eta([b_1]) = \eta((\mathfrak{N}[N_i' : [b_1]/N_i] \setminus [b_1]) : [b_1', ..., b_m'])$.

X-Emp-Application: Assume that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{[M_1, ..., M_q] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : \emptyset\} \cup \Gamma$. Let $t_{(i,j)}$ denote the M_i-N_j-component such that $\eta([M_1, ..., M_q]) = \bigcup_{(i,j)\in\{1..q\}\times\{1..p\}} t_{(i,j)} = \eta([N_1, ..., N_p])$. Choosing $\tau' := \tau \circ \{\mathcal{T}_{(M_i,N_j)} \mapsto t_{(i,j)}\}$, we obtain $\eta = \tau'\mu\nu \circ Sol$, since

$$\begin{aligned} \tau'\mu\nu \circ Sol(M_i) &= \tau'\mu(M_i) = \tau'([\mathcal{T}_{(M_i,N_j)} \mid j \in \{1..p\}]) \\ &= \textstyle\bigcup_{j\in\{1..p\}} t_{(i,j)} = \eta(M_i) \end{aligned}$$

and

$$\begin{aligned} \tau'\mu\nu \circ Sol(N_j) &= \tau'\mu\nu(N_j) = \tau'([\mathcal{T}_{(M_i,N_j)} \mid i \in \{1..q\}]) \\ &= \textstyle\bigcup_{i\in\{1..q\}} t_{(i,j)} = \eta(N_j) \end{aligned}$$

which suffices, as all other variables remain unaffected by the change. It remains to show that η also solves $\mu\nu(\Gamma)$. The assumption that $\eta = \tau'\mu\nu Sol = \tau'\mu\nu Sol \circ \mu\nu$ solves Γ immediately confirms the property of η being a solution also to $\mu\nu(\Gamma)$.

X-Partition: This rule is complete in itself, but in order to prove the completeness of X-Rep-Base later, we conduct a slightly wrong proof. Assume that there exists a substitution τ such that $\eta := \tau \circ Sol$ solves $\{[M_1, M_1, ..., M_r, M_r] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : e\} \cup \Gamma$. In particular, there exist the r-partitions χ of $\eta([N_1, ..., N_p])$ and ζ of e such that for each $i \in \{1..r\}$, $\eta(M_i^{c_i}) = \chi(i) \cup \tau(\zeta(i))$, or as a whole, $\eta([M_1, M_1, ..., M_r, M_r]) = \bigcup_{i\in\{1..r\}}(\chi(i) \cup \tau(\zeta(i))) = \eta([N_1, ..., N_p] : e)$. We now separate an "indefinite" amount β_i off from each $\eta(M_i)$ such that $\chi(i) = \chi'(i) \cup \beta_i^c$ (this ensures that β_i is also a sub-multiset of $\eta([N_1, ..., N_p])$). We further separate β_i into p parts, such that $\beta_i = \bigcup_{j\in\{1..p\}} t_{i,j}$ where for each j, $t_{i,j}$ comes from $\eta(N_j)$ (i.e. $\gamma_j := \bigcup_{i\in[1,1..r]} t_{i,j}$ with $\eta(N_j) = n_j \cup \gamma_j$). We call $t := [t_{i,j} \mid i \in [1, 1..r], j \in \{1..p\}]$. We would obtain the "correct" proof for X-Partition by setting $t_{i,j} = \emptyset$ for all i and j. To be shown is the existence of a substitution τ' such that $\eta = \tau'\mu \circ Sol$ solves $\{[\mathcal{M}_{\alpha_i}^{c_i}] \stackrel{?}{=} \mathbf{N}_i : \zeta(i) \mid_{i\in\{1..r\}}\} \cup \{[\mathcal{N}_1..\mathcal{N}_p] \stackrel{?}{=} [\mathbf{N}_1..\mathbf{N}_r]\} \cup \mu(\Gamma)$. Choosing $\tau' := \tau \circ \{\mathbf{N}_i \mapsto \chi'(i), \mathcal{M}_{\alpha_i} \mapsto \tau(M_i) \mid_{i\in\{1..r\}}\} \circ \{\mathcal{N}_j \mapsto n_j \mid_{j\in\{1..p\}}\} \circ \{\mathcal{T}_{(M_i,N_j)} \mapsto t_{i,j} \mid_{(i,j)\in\{1..r\}\times\{1..p\}}\}$, we obtain $\eta = \tau'\mu \circ Sol$, since

$$\begin{aligned} \tau'\mu \circ Sol(M_i) &= \tau'\mu(M_i) = \tau'(\mathcal{M}_{\alpha_i}) \\ &= \tau(M_i) = \tau \circ Sol(M_i) \end{aligned}$$

as well as

$$\begin{aligned} \tau'\mu \circ Sol(N_j) &= \tau'\mu(N_j) = \tau'(\mathcal{N}_j \cup [\mathcal{T}_{(M_i,N_j)} \mid_{i\in[1,1..r]}]) \\ &= n_j \cup \gamma_j = \tau \circ Sol(N_j) \end{aligned}$$

which suffices as all other variables are not affected by the change. It remains to show that η also solves $\{[\mathcal{M}_{\alpha_i}^{c_i}] \stackrel{?}{=} \mathbf{N}_i : \zeta(i) \mid_{i\in\{1..r\}}\} \cup \{[\mathcal{N}_1..\mathcal{N}_p] \stackrel{?}{=} [\mathbf{N}_1..\mathbf{N}_r]\} \cup \mu(\Gamma)$. η

solves $\mu(\Gamma)$, as $\eta = \eta\mu$ already solved Γ. Applying η on the first r equations, we get $\eta([\mathcal{M}^{c_i}_{\alpha_i}]) = \tau(M^{c_i}_i) \stackrel{?}{=} \chi'(i) \cup \tau(\zeta(i)) = \eta(\mathbf{N}_i : \zeta(i))$ for each i, which is *almost* unified: we are missing $\beta^{c_i}_i$ on the right-hand side, which we will compensate for in X-Rep-Base. Applying η on the $(r+1)$-th equation, we get $\eta([\mathcal{N}_1..\mathcal{N}_p]) = \bigcup_{j\in\{1..p\}} n_j \stackrel{?}{=} \tau([N_1, ..., N_p]) \setminus t = \bigcup_{i\in\{1..r\}} \chi'(i) = \tau([\mathbf{N}_1..\mathbf{N}_r])$ is already unified.

X-Rep-Distribution: Assume that there exists a substitution τ such that $\eta := \tau \circ \mathit{Sol}$ solves $\{[\mathcal{M}^c] : \emptyset \stackrel{?}{=} \mathbf{N} : [b_1, ..., b_n]\} \cup \Gamma$. To be shown is the existence of a substitution τ' such that $\eta = \tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ \mathit{Sol}$ solves $\{[(\mathcal{M}')^c] : [b_1^{c-1}] \stackrel{?}{=} \mathbf{N} : [b_2, ..., b_n]\} \cup \Gamma[\mathcal{M}' : [b_1]/\mathcal{M}]$. Choosing $\tau' := \tau \circ \{\mathcal{M}' \mapsto \tau(\mathcal{M} \setminus [b_1])\}$, for the only variable the change could have an effect,

$$\begin{aligned}\tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ \mathit{Sol}(\mathcal{M}) &= \tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\}(\mathcal{M}) \\ &= \tau'(\mathcal{M}' : [b_1]) \\ &= \tau(\mathcal{M} \setminus [b_1] : [b_1]) \\ &= \tau(\mathcal{M}) = \tau \circ \mathit{Sol}(\mathcal{M})\end{aligned}$$

showing $\eta = \tau' \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\} \circ \mathit{Sol}$. It remains to show that η also solves $\{[(\mathcal{M}')^c] : [b_1^{c-1}] \stackrel{?}{=} \mathbf{N} : [b_2, ..., b_n]\} \cup \Gamma[\mathcal{M}' : [b_1]/\mathcal{M}]$. η solves $\Gamma[\mathcal{M}' : [b_1]/\mathcal{M}]$ as it already solved Γ and $\eta = \eta \circ \{\mathcal{M} \mapsto \mathcal{M}' : [b_1]\}$. Applying η on the left-hand side of $\cup$ gives us an equation we get from the assumption, with the only difference that a $[b_1]$ is missing from both sides of $\stackrel{?}{=}$, which does not interfere with our goal, namely that η remains a solution.

X-Rep-Base: This rule is not complete in itself, but only in combination with X-Partition. Assume that there exists a substitution τ such that $\eta := \tau \circ \mathit{Sol}$ solves $\{[\mathcal{M}^c_\alpha] : e \stackrel{?}{=} \mathbf{N} : \emptyset\} \cup \Gamma$. In particular, $\eta(\mathcal{M}^c_\alpha) \cup \eta(e) = \eta(\mathbf{N})$. We dub $\beta := \eta(\mathcal{M}_\alpha)$, which is the "indefinite amount" we addressed in the completeness proof for X-Partition. To be shown is the existence of a substitution τ' such that $\eta = \tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ \mathit{Sol}$ unifies $\Gamma[\alpha/\mathcal{M}_\alpha,\ e/\mathbf{N}]$. We can choose τ' in such a way that $\tau'(\alpha) = \beta$ and $\tau'(x) = \tau(x)$ otherwise, because either α is a non-empty multiset of set variables, or α is empty, in which case p in X-Partition must have been 0 such that β, a sub-multiset of $\eta([N_1, ..., N_p])$, must also be empty. Now $\eta = \tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ \mathit{Sol}$ almost holds, since

$$\begin{aligned}\tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ \mathit{Sol}(\mathbf{N}) &= \tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\}(\mathbf{N}) \\ &= \tau'(e) = \tau(e) \\ &\approx \tau(e) \cup \beta^c = \tau \circ \mathit{Sol}(\mathbf{N})\end{aligned}$$

but the extra β^c is exactly what we were missing in X-Partition, and

$$\begin{aligned}\tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\} \circ \mathit{Sol}(\mathcal{M}_\alpha) &= \tau' \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\}(\mathcal{M}_\alpha) \\ &= \tau'(\alpha) = \beta = \tau \circ \mathit{Sol}(\mathcal{M}_\alpha).\end{aligned}$$

It remains to show that η also solves $\Gamma[\alpha/\mathcal{M}_\alpha,\ e/\mathbf{N}]$, which holds due to the assumption that η solved Γ already and $\eta = \eta \circ \{\mathcal{M}_\alpha \mapsto \alpha,\ \mathbf{N} \mapsto e\}$.

$$\text{X-ORIENTATION}\quad \frac{(Sol, \{\mathfrak{M}_1 : e_1 \stackrel{?}{=} \mathfrak{M}_2 : \emptyset\} \cup \Gamma)}{(Sol, \{\mathfrak{M}_2 : \emptyset \stackrel{?}{=} \mathfrak{M}_1 : e_1\} \cup \Gamma)} e_1 \neq \emptyset \qquad \text{X-CLASH}\quad \frac{(Sol, \emptyset \stackrel{?}{=} \mathfrak{M} : e)}{Fail} e \neq \emptyset$$

$$\text{X-DISTRIBUTION}\quad \frac{(Sol, \{\mathfrak{M} : [b_1, ..., b_k] \stackrel{?}{=} [N_1, ..., N_p] : [b'_1, ..., b'_m]\} \cup \Gamma)}{\begin{array}{c} |_{i=1}^{k} (Sol, \{b_1 \stackrel{?}{=} b'_i, \mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} [N_1, ..., N_p] : [b'_1, ..., b'_{i-1}, b'_{i+1}, ..., b'_m]\} \cup \Gamma) \\ |_{i=1}^{p} (\nu \circ Sol, \{\mathfrak{M} : [b_2, ..., b_k] \stackrel{?}{=} (\nu([N_1, ..., N'_i, ..., N_p])) : [b'_1, ..., b'_m]\} \cup \Gamma) \\ \text{where } N'_i \text{ is fresh and } \nu = \{N_i \mapsto N'_i : [b_1]\} \end{array}} {\scriptstyle k>0, \atop m>0}$$

$$\text{X-APPLICATION}\quad \frac{(Sol, \{[M_1, ..., M_q] : \emptyset \stackrel{?}{=} [N_1, ..., N_p] : e\} \cup \Gamma)}{\begin{array}{c} |_{\zeta \in Z(e,r)} (\mu_\zeta \circ \nu \circ Sol, \Gamma) \\ \text{where } \mu_\zeta = \{M_i \mapsto [\mathcal{T}_{(M_i,N_j)} \mid j \in \{1..p\}] : \zeta(i) \mid i \in \{1..q\}\} \\ \text{and } \nu = \{N_j \mapsto [\mathcal{T}_{(M_i,N_j)} \mid i \in \{1..q\}] \mid j \in \{1..p\}\} \end{array}} q > 0, e \neq \emptyset$$

Figure 3.7: Algorithm for the linear case

3.4.4 Linearity restriction

If the set variables are linear, i.e. any set variable occurs at most once in the whole problem, the rules can be simplified to those shown in Figure 3.7. X-SEMI-TAUTOLOGY becomes unnecessary, as the condition $\mathfrak{M} \cap \mathfrak{N} \neq \emptyset$ will never be met. Instead of partitioning the problem with X-PARTITION, X-APPLICATION finds the distribution directly. In X-DISTRIBUTION and X-APPLICATION, set variables in Γ do not need to be substituted away, as they only occur in the equation that is currently being observed.

3.5 Single chain on one side

3.5.1 Problem statement

A chain variable $\mathbf{Ch}_i$ ($i \in \mathbb{N}_0$) can be instantiated by a function of the type $Var \to Var \to Expr$ where for any $a, b \in Var$, $\mathbf{Ch}_i(a, b)$ is a chain expression, i.e. an expression of the form $[a = x_1, x_1 = x_2, ..., x_{n-1} = x_n, x_n = b]$. The function could be written as $\lambda ab.[a = x_1, x_1 = x_2, ..., x_{n-1} = x_n, x_n = b]$, following the style of the lambda calculus, or as $[\,\cdot = x_1, x_1 = x_2, ..., x_{n-1} = x_n, x_n = \cdot\,]$ with $\cdot$ representing the "holes" which the argument variables can be inserted into. In the problem, the variable always occurs in the form $\mathbf{Ch}_i(a, b)$. For instance, the problem

$$\mathbf{Ch}_1(A, \mathsf{b}) \doteq [\mathsf{a} = \mathsf{x}, \mathsf{x} = \mathsf{b}],\ \mathbf{Ch}_1(\mathsf{c}, \mathsf{d}) \doteq [C = \mathsf{x}, X = \mathsf{d}] \tag{3.1}$$

can be solved by $\{\mathbf{Ch}_1 \mapsto [\,\cdot = \mathsf{x}, \mathsf{x} = \cdot\,],\ A \mapsto \mathsf{a},\ C \mapsto \mathsf{c},\ X \mapsto \mathsf{x}\}$.

A chain is not allowed to have any circles, i.e., all the variables left of the bindings have to be distinct: The "punched" expression $[\cdot = b, b = c, c = b, b = \cdot]$ cannot be instantiated into a valid chain, because "$b =$" occurs twice. $[\cdot = b, b = \cdot]$ could be valid, but instantiated with (b, x), the resulting expression $[b = b, b = x]$ is not a valid chain, as, again, "$b =$" occurs twice. The exclusion of cycles comes with gains in expressiveness. For instance, "$X \neq Y$" can be encoded by the equation $\mathbf{Ch}(\mathsf{a}, \mathsf{b}) \doteq [\mathsf{a} = X, X = Y, Y = \mathsf{b}]$.

We restrict the extension to contain only one set variable per equation, analogous to the restriction in Section 3.1, but additionally demanding the variables to be linear, i.e. each variable can only occur once in the problem (as opposed to the example (3.1)).

As another extension, one could consider allowing one variable on both sides of equations like in Section 3.2 or 3.3. However, this extension would not be closed, at least if we try to follow the same procedure as in the previous sections: given a problem $\mathbf{Ch}_1 : e \doteq \mathbf{Ch}_2 : e'$, when we try to distribute e over to the right-hand side to empty the bindings on the left-hand side, $b \in e$ could occur anywhere in $\mathbf{Ch}_2$. Thus, we have to split $\mathbf{Ch}_2$ in two, obtaining $\mathbf{Ch}_1 : (e \setminus [b]) \doteq [\mathbf{Ch}_2', \mathbf{Ch}_2''] : e'$, which is out of the restriction.

3.5.2 Solution

Reusing the rules from the simple unification problem as well as E-Set-Distribution, E-Clash and E-Set-Orientation from Section 3.1, any one-sidedly chain-extended problem can be reduced to a problem containing only equations of the form $\mathbf{Ch}(a, b) \doteq e$, where e is a simple expression. As the length of e is known, the problem can be solved by $\mathbf{Ch} \mapsto [\cdot = X_1, X_1 = X_2, ..., X_{|e|-1} = \cdot]$ (X_i are fresh) and using simple algorithm for the rest.

However, that alone would leave us in trouble as, if a chain variable is mapped to $[\cdot = X, X = \mathsf{b}, \mathsf{b} = \cdot]$, the prohibition of cycles in the chain requires us to remember that any branch that maps X to b must be discarded, should the necessity arise later in the process. Furthermore, if the algorithm terminates, still containing the meta variable, the solution must include this constraint.

To this end, we introduce the new construct of *duplicate-avoidance-sets* into the *Solver* data structure. When substituting $\mathbf{Ch}(a, b)$ with the fresh X_i, we add $\{a, X_1, ..., X_{|e|-1}\}_{\mathcal{D}?}$ to Γ, which takes part in all substitutions applied on Γ and acts according to the rules shown in Figure 3.8. As soon as two variables in the set are the same, the branch fails (DA-Crash); if the whole set only consists of distinct ground variables, the set is discarded. The algorithm terminates (DA-Termination) when Γ contains no more reducible equations, leaving only the (empty or non-empty) final constraints for the meta variables.

3.5.3 Correctness

With the variables being linear, the correctness is straightforward enough we allow ourselves to omit a formal proof at this point. May we point out that linearity is in fact

DA-Crash

$$\frac{(Sol, \{[..., x_i, ..., x_i, ...]_{\mathcal{D}?}\} \cup \Gamma)}{\mathit{Fail}}$$

DA-Tautology

$$\frac{(Sol, \{A_{\mathcal{D}?}\} \cup \Gamma)}{(Sol, \Gamma)} \quad \text{A is distinct and contains only ground variables}$$

DA-Termination

$$\frac{(Sol, \Gamma)}{(Sol, \emptyset)} \quad \text{Γ contains only distinct DA-multisets}$$

SCh-Application

$$\frac{(Sol, \{\mathbf{Ch}_i(a, z) : \emptyset \stackrel{?}{=} e\} \cup \Gamma)}{(\{\mathbf{Ch}_i \mapsto [\cdot = X_1, ..., X_{|e|-1} = \cdot]\} \circ Sol, \{\{a, X_1, ..., X_{|e|-1}\}_{\mathcal{D}?}\} \cup \Gamma[[\cdot = .. = \cdot]/\mathbf{Ch}_i])}$$

SCh-Empty-Crash

$$\frac{(Sol, \{\mathbf{Ch}(a, z) : \emptyset \stackrel{?}{=} \emptyset\} \cup \Gamma)}{\mathit{Fail}}$$

Figure 3.8: Algorithm for the one-sidedly chain-extended problem

needed: given a problem $\mathbf{Ch}(a, b) = [..]$, $\mathbf{Ch}(x, y) = [..]$, we would substitute $\mathbf{Ch}$ by $[\cdot = X_1, ..., X_n = \cdot]$, yielding $[a = X_1, ..., X_n = b] = [..]$, $[x = X_1, ..., X_n = y] = [..]$. But only remembering $\{a, X_1, ...\}$ to be distinct, we fail to prohibit $X_3 \mapsto x$, which introduces a cycle into the second chain.

4 Implementation

The fully multiset-extended problem covered in Section 3.4 was implemented as a stack-project [11], which can be found under `https://github.com/1qxj-yt/kettenunif`. After cloning the git repository, running `stack build` inside the folder installs all the dependencies needed and builds the application. `stack run` starts the application.

4.1 Functionalities

The application prompts a welcome message, whereupon it enters a read-eval-print-loop (REPL):

```
Welcome!
Type a unifcation problem, :v to toggle verbosity or :q to quit.
> _
```

There are four types of commands accepted by the interpreter.

$$\langle \mathit{Command} \rangle ::= \langle \mathit{Control} \rangle \mid \langle \mathit{UnifProb} \rangle \mid \langle \mathit{SubstApp} \rangle \mid \langle \mathit{SubstComp} \rangle$$

Control Commands

"`:q`" quits the application.
"`:v`" toggles the verbosity of the problem solver. The verbosity modes will be explained in the next subsubsection.

$$\langle \mathit{Control} \rangle ::= \texttt{:q} \mid \texttt{:v}$$

Unification Problem

The elements of the problem set (*UnifProb*) are separated by a comma (`,`). For each problem element (*ProbEl*), the two expressions to be unified are separated by an equals-sign followed by a full-stop (`=.`). An expression consists of two parts: the set-variable part and the bindings part. If the set part is non-empty, the two parts are separated by a colon (`:`). The set part is a finite list of set variables separated by semicolons (`;`). A set variable is an "`M`" followed by a natural number, e.g. "`M21`". If no number is specified (`M`), it will be interpreted as "`M0`". The set variable can be followed by a finite amount of apostrophes (`'`), in which case the variable is interpreted as a helper variable. The bindings part is a finite list of bindings enclosed in square brackets (`[]`) and separated by commas (`,`). A binding is a pair of variables separated by an equals-sign (`=`), where a variable is a latin letter followed by a number. If no number is specified (`x`), it will be interpreted as being followed by a zero (`x0`). If the letter is small, the variable is ground; if it is capital, the variable is meta.

The syntax is summarized in the following grammar:

$$
\begin{aligned}
\langle \mathit{UnifProb} \rangle &::= \varepsilon \mid \langle \mathit{ProbEl} \rangle \mid \langle \mathit{ProbEl} \rangle \texttt{,} \langle \mathit{UnifProb} \rangle \\
\langle \mathit{ProbEl} \rangle &::= \langle \mathit{Expr} \rangle \texttt{=.} \langle \mathit{Expr} \rangle \\
\langle \mathit{Expr} \rangle &::= \texttt{[} \langle \mathit{Binds} \rangle \texttt{]} \mid \langle \mathit{Sets} \rangle \texttt{:[} \langle \mathit{Binds} \rangle \texttt{]} \\
\langle \mathit{Sets} \rangle &::= \langle \mathit{SetVar} \rangle \mid \langle \mathit{SetVar} \rangle \texttt{;} \langle \mathit{Sets} \rangle \\
\langle \mathit{SetVar} \rangle &::= \texttt{M} \langle \mathit{Int} \rangle \langle \mathit{Apos} \rangle \\
\langle \mathit{Apos} \rangle &::= \varepsilon \mid \texttt{'} \mid \texttt{'} \langle \mathit{Apos} \rangle \\
\langle \mathit{Binds} \rangle &::= \varepsilon \mid \langle \mathit{Bind} \rangle \mid \langle \mathit{Bind} \rangle \texttt{,} \langle \mathit{Binds} \rangle \\
\langle \mathit{Bind} \rangle &::= \langle \mathit{Var} \rangle \texttt{=} \langle \mathit{Var} \rangle \\
\langle \mathit{Var} \rangle &::= \langle \mathit{GroundVar} \rangle \mid \langle \mathit{MetaVar} \rangle \\
\langle \mathit{GroundVar} \rangle &::= \langle \mathit{UpperChar} \rangle \langle \mathit{Int} \rangle \\
\langle \mathit{MetaVar} \rangle &::= \langle \mathit{LowerChar} \rangle \langle \mathit{Int} \rangle \\
\langle \mathit{Int} \rangle &::= \varepsilon \mid \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \ldots \\
\langle \mathit{UpperChar} \rangle &::= \texttt{A} \mid \texttt{B} \mid \texttt{C} \mid \texttt{D} \mid \ldots \mid \texttt{Y} \mid \texttt{Z} \\
\langle \mathit{LowerChar} \rangle &::= \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{d} \mid \ldots \mid \texttt{y} \mid \texttt{z}
\end{aligned}
$$

The behaviour following the input of a unification problem depends on the current verbosity mode, of which there are three. In `Silent`-mode, all the solutions will be listed:

```
> [X = a, B = C] =. M2;M2:[X = X3, A = x], [X = g0, H8 = s] =. M:[b = g]
[{M→[H8=s],M2→[]|B→A,C→x,X→b,X3→a}]
```

whereas in `Counting`-mode, only the number of solutions will be displayed, as well as the proportion of distinct solutions to the number of all solutions in the list:

```
> :v
Switched verbosity to: Count
> [X = a, B = C] =. M2;M2:[X = X3, A = x], [X = g0, H8 = s] =. M:[b = g]
1 [1/1=100%]
```

In `Verbose`-mode, the whole derivation tree will be printed from which the solutions were obtained:

```
> :v
Switched verbosity to: Verbose
> [X = a, B = C] =. M2;M2:[X = X3, A = x], [X = g0, H8 = s] =. M:[b = g]
(id, E [X=a,B=C] :=?: E M2;M2:[X=X3,A=x] ∪ Γ)
x-distribution
```

```
(id, B X=a :=?: B X=X3 ∪ Γ)
decomposition
  (id, V a :=?: V X3 ∪ Γ)
  orientation
    (id, V X :=?: V X ∪ Γ)
    tautology
      (id, V X3 :=?: V a ∪ Γ)
      application
        ({id|X3→a}, E [B=C] :=?: E M2;M2:[A=x] ∪ Γ)
        x-distribution
```

⋮

Substitution applied on Expressions

A substitution application is a finite list of substitutions followed by an expression. A substitution is enclosed in curly brackets ({}) and consists of two components: the set-variable component and the variable component. If the set-variable component is non-empty, the two components are separated by a vertical line (|). The set component is a finite list of set-maps separated by commas. A set-map is a set variable followed by a hyphen-minus (-), a greater-than sign (>) and an expression. The variable component is a finite list of variable maps separated by commas, where a variable-map is a meta variable followed by hyphen-minus, a greater-than sign and a (ground or meta) variable.

The syntax is summarized in the following grammar:

$$
\begin{aligned}
\langle \mathit{SubstApp} \rangle &::= \langle \mathit{SubstList} \rangle \langle \mathit{Expr} \rangle \\
\langle \mathit{SubstList} \rangle &::= \langle \mathit{Subst} \rangle \mid \langle \mathit{Subst} \rangle \langle \mathit{SubstList} \rangle \\
\langle \mathit{Subst} \rangle &::= \texttt{\{}\langle \mathit{VarComp} \rangle\texttt{\}} \mid \texttt{\{}\langle \mathit{SetComp} \rangle \;\texttt{|}\; \langle \mathit{VarComp} \rangle\texttt{\}} \\
\langle \mathit{VarComp} \rangle &::= \varepsilon \mid \langle \mathit{VarMap} \rangle \mid \langle \mathit{VarMap} \rangle \texttt{,} \langle \mathit{VarComp} \rangle \\
\langle \mathit{VarMap} \rangle &::= \langle \mathit{MetaVar} \rangle \texttt{->} \langle \mathit{Var} \rangle \\
\langle \mathit{SetComp} \rangle &::= \langle \mathit{SetMap} \rangle \mid \langle \mathit{SetMap} \rangle \texttt{,} \langle \mathit{SetComp} \rangle \\
\langle \mathit{SetMap} \rangle &::= \langle \mathit{SetVar} \rangle \texttt{->} \langle \mathit{Expr} \rangle
\end{aligned}
$$

The result of running the command is the input expression with the substitutions applied on it successively from right to left.

```
> {X -> a, B -> C, Y -> a} [X = x, B = C]
E [a=x,C=C]
> {C -> c} {X -> a, B -> C, Y -> a} [X = x, B = C]
E [a=x,c=c]
> {M1 -> M2:[] | } M1:[X = x, B = C]
E M2:[a=x,B=C]
> {M1 -> M2:[] | X -> a } [X = x, B = C]
E [a=x,B=C]
```

Substitution Composition

A substitution composition is a finite list of substitutions.

$$\langle \mathit{SubstComp} \rangle ::= \langle \mathit{SubstList} \rangle$$

The result is the composition of substitutions from right to left:

```
> {X -> a, B -> C, Y -> a} {C -> B, B -> X}
{id|B→a,X→a,Y→a}
> {M1 -> M2:[] |}  {M0 -> M1:[] |}
{M→M2:[],M1→M2:[]|id}
```

The resulting substitution is restricted (Def. 2.1.10) to non-helper variables:

```
> {M1' -> M2:[] |}  {M0 -> M1':[] |}
{M→M2:[]|id}
```

4.2 Accelerating rules

The following rules are not needed for correctness, but aim to slightly accelerate the whole process:

$$\textsc{X-AccelL}\quad \frac{(\mathit{Sol}, \{M : \emptyset \stackrel{?}{=} \mathfrak{N} : e\} \cup \Gamma)}{(\{M \mapsto \mathfrak{N} : e\} \circ \mathit{Sol}, \Gamma[\mathfrak{N} : e/M])}\text{eqn. is non-block}$$

$$\textsc{X-AccelR}\quad \frac{(\mathit{Sol}, \{\mathfrak{M} : e \stackrel{?}{=} N : \emptyset\} \cup \Gamma)}{(\{N \mapsto \mathfrak{M} : e\} \circ \mathit{Sol}, \Gamma[\mathfrak{M} : e/N])}\text{eqn. is non-block}$$

Also, it might be beneficial to use E-Tautolgy when applicable, which is a special case of X-Emp-Application.

4.3 Tests

Apart from the proofs provided in the previous chapter, the soundness of the algorithm was also experimentally examined through QuickCheck[1] version 2.9.2. Random problems were created and the solutions checked, i.e. each of the solutions were applied onto both sides of every equation in the problem, and their equality was verified.

To ensure a reasonably quick but meaningful verification during development, the following restrictions were placed upon the variable space as well as the length of the expressions: The variables were chosen from a linear distribution on the set $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{x}, \mathsf{y}, \mathsf{z},$

$A, B, X, Y\}$, as well as the set variables from $\{M_1, ..., M_{10}\}$. The lengths of the variable-component and the set-variable-component of an expression were chosen from the range from 0 to 3, respectively.

To inspect the randomization directly, move to the `test` folder, start `ghci` and load the module `Simple.SoundnessAuto.Bi_Mset`:

```
machine@user kettenunif % cd test
machine@user test % stack -- exec ghci
GHCi, version x.y.z: http://www.haskell.org/ghc/  :? for help
Prelude> :l Simple.SoundnessAuto.Bi_Mset
[1 of 1] Compiling Simple.SoundnessAuto.Bi_Mset
    ( Simple/SoundnessAuto/Bi_Mset.hs, interpreted )
Ok, modules loaded: Simple.SoundnessAuto.Bi_Mset.
```

Then, typing `generate arbitrary` with its type specified prints a random value from the distribution described above.

```
*Simple.SoundnessAuto.Bi_Mset> generate arbitrary :: IO Var
X
*Simple.SoundnessAuto.Bi_Mset> generate arbitrary :: IO Var
Y
*Simple.SoundnessAuto.Bi_Mset> generate arbitrary :: IO Bind
B=A
*Simple.SoundnessAuto.Bi_Mset> generate arbitrary :: IO Bind
A=A
*Simple.SoundnessAuto.Bi_Mset> generate arbitrary :: IO UnifProblemEl
M10;M1:[x=z,B=a] =. M9;M10;M7:[y=X,B=Y]
```

To generate samples on a particular seed, load the QuickCheck `Random` and `Gen` modules. (Typing `:set prompt "*> "` hides the module name(s)).

```
*> :m +Test.QuickCheck.Random Test.QuickCheck.Gen
```

Now, when g is the seed and s the size of the sample, `unGen arbitrary (mkQCGen` g`)` s with its type specified yields constant results, as long as g and s stay the same. The size has no effect on the relevant types, except for the unification problem, which is (for the purpose of observing the random generation) a list (actually, a set) of unification problem elements. In that case, the size limits the maximal length of the list.

```
*> unGen arbitrary (mkQCGen 42) 0 :: Var
c
*> unGen arbitrary (mkQCGen 42) 0 :: Var
c
*> unGen arbitrary (mkQCGen 42) 0 :: [UnifProblemEl]
[]
*> unGen arbitrary (mkQCGen 42) 1 :: [UnifProblemEl]
```

```
[]
*> unGen arbitrary (mkQCGen 42) 2 :: [UnifProblemEl]
[M9;M1:[z=x] =. [z=y,x=a,A=X]]
```

Note that the QuickCheck version can influence the seed-to-sample relation, such that the behaviour might not be reproducible in different versions.

Instead of relying solely on randomness, there were also concrete test cases implemented:

input problem	expected output
$\{\mathsf{x} = Y \doteq X = \mathsf{y}\}$	$[\{X \mapsto \mathsf{x}, Y \mapsto \mathsf{y}\}]$
$\{\mathsf{x} = \mathsf{x} \doteq \mathsf{z} = \mathsf{z}\}$	$[\,]$
$\{X = Y \doteq Y = \mathsf{a}\}$	$[\{X \mapsto \mathsf{a}, Y \mapsto \mathsf{a}\}]$
$\{X = Y \doteq Y = A\}$	$[\{X \mapsto A, Y \mapsto A\}]$
$\{X = Y \doteq Y = A\}$	$[\{A \mapsto X, Y \mapsto X\}]$
$\{[A = B, C = D] \doteq [\mathsf{x} = \mathsf{y}, \mathsf{z} = \mathsf{w}]\}$	$[\{A \mapsto \mathsf{x}, B \mapsto \mathsf{y}, C \mapsto \mathsf{z}, D \mapsto \mathsf{w}\},$ $\{A \mapsto \mathsf{z}, B \mapsto \mathsf{w}, C \mapsto \mathsf{x}, D \mapsto \mathsf{y}\}]$
$\{M : [X = \mathsf{a}] \doteq [A = \mathsf{a}, B = D]\}$	$[\{M \mapsto [B = D] \mid X \mapsto A\},$ $\{M \mapsto [A = \mathsf{a}] \mid D \mapsto \mathsf{a}, X \mapsto B\}]$
$\{[M, M] : \emptyset \doteq [A = \mathsf{a}, \mathsf{a} = \mathsf{a}]\}$	$[\{M \mapsto [\mathsf{a} = \mathsf{a}] \vert A \mapsto \mathsf{a}\}]$

When checking the solutions, the substitutions should be compared for equivalence on the variables that appear in the problem, instead for exact equality. For instance, for the problem $\{X = Y \doteq Y = A\}$, the substitutions $\{A \mapsto X, Y \mapsto X\}$ and $\{A \mapsto Y, X \mapsto Y\}$ are just renamings of each other and thus equally appropriate (and general, checking also a small proportion of completeness).

On top of the tests on unification problems, there were also concrete test cases imposed on substitution application as well as composition. Furthermore, the property enforced on the ordering on unification problem elements in Subsection 3.2.2 is tested on random samples. Running `stack test` in the top folder runs all these tests, which, as of January 2020, all pass successfully, after having repeatedly helped during development by failing correctly.

5 Conclusion

After motivating the unification problem of letrec-bindings through the proof of correct program transformations, a solution to the simple version was proven terminating, sound and complete. In Chapter 3, multiset extensions were successively generalized, beginning with restricted cases heavily limiting occurrences of set variables. Allowing set or multiset variables on both sides of the equation required explicit specifications on the order of rule applications to be established for the process to terminate. The fully multiset-extended problem, which allows an arbitrary number of possibly the same multiset variable at any place in the problem, was constructively shown decidable, which was previously unknown. Unlike in the previous extensions, making every rule sound and complete would have forced the algorithm into non-termination, wherefore two rules that were not sound and complete on their own cancelled out each other's "coordinated misconduct". As a special type of set variable, chain variables were also briefly discussed. In Chapter 4, a REPL implementation of the algorithm was presented, along with tests imposed on the program.

Outlook

There remains potential for further improvement and development. For example, the only solution to $\{[A = B, A = B] \doteq [\mathsf{a} = \mathsf{b}, \mathsf{a} = \mathsf{b}]\}$ is $\{A \mapsto \mathsf{a}, B \mapsto \mathsf{b}\}$, but the algorithm finds two times the same solution:

```
> [A=B,A=B] =. [a=b,a=b]
[{id|A→a,B→b},{id|A→a,B→b}]
```

If the solutions are *exactly* the same, counting-mode finds those overlaps:

```
> :v
Switched verbosity to: Count
> [A=B,A=B] =. [a=b,a=b]
2 [1/2=50%]
```

As the problem, and with it, the size of the solution set, becomes larger, those exact overlaps can get unpleasant:

```
> M10:[Y=X,x=b] =. M8;M9:[A=z,X=Y,A=b] ,
      M2;M8:[] =. [x=B,a=b,A=X],
      M10:[x=z,B=x] =. M9;M9:[X=B,B=X,A=x]
434 [40/434=9%]
```

If the duplicates are not exact, counting-mode does not see that a solution could be renamed into another: The set of solutions to $\{M_0 : [\mathsf{a} = \mathsf{a}, \mathsf{a} = \mathsf{a}] \doteq M_1 : [\mathsf{a} = \mathsf{a}]\}$ can be expressed as $[\{M_0 \mapsto T : \emptyset, M_1 \mapsto T : [\mathsf{a} = \mathsf{a}]\}]$, but the algorithm finds three different solutions:

```
> M:[a=a,a=a] =. M1:[a=a]
3 [3/3=100%]
```

More precisely:

```
> :v
Switched verbosity to: Verbose
> M:[a=a,a=a] =. M1:[a=a]
[
    {M→T(M1,M):[],      M1→T(M1,M):[a=a]       |id},
    {M→T(M,M1'):[],     M1→T(M,M1'):[a=a]      |id},
    {M→T(M,M1''):[a=a],M1→T(M,M1''):[a=a,a=a]|id}
]
```

A fix has to involve at least X-Distribution, in which the equivalence of spawned branches could be recognized.

Another direction in which further progress could be made is the treatment of chain variables, which remained very limited in this thesis. For cases like

$$[\mathbf{Ch}(a, b), \mathbf{Ch}(c, d)] \stackrel{?}{=} [\mathbf{Ch}(x, y), \mathbf{Ch}(z, w)],$$

we have to begin by clarifying what kind of solutions we even want a potential algorithm to provide us with.

Bibliography

[1] Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ICFP '00, pages 268–279, New York, NY, USA, 2000. ACM.

[2] Agostino Dovier, Alberto Policriti, and Gianfranco Rossi. Integrating lists, multisets, and sets in a logic programming framework. In Franz Baader and Klaus U. Schulz, editors, *Frontiers of Combining Systems: First International Workshop, Munich, March 1996*, pages 303–319, Dordrecht, 1996. Springer Netherlands.

[3] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, April 1982.

[4] Zan Naeem and Giselle Reis. Unification of multisets with multiple labelled multiset variables. In *Informal Proceedings of UNIF 2019, 33rd International Workshop on Unification*, 2019.

[5] Will Partain, André Santos, and Simon Peyton Jones. Let-floating: moving bindings to give faster programs, May 1996. ACM SIGPLAN International Conference on Functional Programming (ICFP'96).

[6] M.S. Paterson and M.N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158 – 167, 1978.

[7] Benjamin C. Pierce. *Types and Programming Languages*, chapter 3.5. The MIT Press, 2002.

[8] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.

[9] David Sabel. Automating the diagram method to prove correctness of program transformations. *Electronic Proceedings in Theoretical Computer Science*, 289:17–33, 02 2019.

[10] Manfred Schmidt-Schauß and David Sabel. Unification of program expressions with recursive bindings. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*, PPDP '16, pages 160–173, New York, NY, USA, 2016. ACM.

[11] The haskell tool stack. `https://docs.haskellstack.org/en/stable/README/`, 2019. Accessed: November 6, 2019.